

DTIC FILE COPY

AD-A216 279



A 3-D VIRTUAL ENVIRONMENT
DISPLAY SYSTEM

THESIS

Robert Edward Filer
Captain, USAF

AFIT/GCS/ENG/89D-2

DTIC
ELECTE
DEC 15 1989
S B

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

89 12 15 043

AFIT/GCS/ENG/89D-2

A 3-D VIRTUAL ENVIRONMENT
DISPLAY SYSTEM

THESIS

Robert Edward Filer
Captain, USAF

AFIT/GCS/ENG/89D-2

DTIC
ELECTE
DEC 15 1989
S B D

Approved for public release; distribution unlimited

AFIT/GCS/ENG/89D-2

A 3-D VIRTUAL ENVIRONMENT
DISPLAY SYSTEM

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science (Computer Systems)

Robert Edward Filer, B.S.
Captain, USAF

December, 1989

Approved for public release; distribution unlimited

Preface

The purpose of this study was to research the current state-of-the-art in Virtual Environment Display Systems and to develop such a system at the Air Force Institute of Technology (AFIT). This involved the development of a software library and a head-mounted display (HMD).

I am indebted to many individuals for their assistance on this project. First, I would like to thank my advisor, Major Phil Amburn, who was always there to answer my questions and put me back on the right track. I want to thank the AFIT Model Shop, especially Mr. Jan LeValley, for the construction of the AFIT Head Mounted Display II. My gratitude also goes to Armstrong Aerospace Medical Research Laboratory, the National Oceanic and Atmospheric Administration, and the Air Force Office of Scientific Research for their sponsorship of this effort.

I would finally like to thank my wife Sherrie for hanging in there and toughing it out with me.

Robert Edward Filer

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	



Table of Contents

	Page
Preface	ii
List of Figures	vi
List of Tables	viii
Abstract	ix
I. Introduction	1
1.1 Background	1
1.2 Problem	1
1.3 Scope	2
1.4 Assumptions	2
1.4.1 Input Server Hardware Selection.	4
1.4.2 Graphics Hardware Selection.	4
1.4.3 Programming Language Selection.	5
1.5 Literature Review	5
1.5.1 University of Utah.	6
1.5.2 Human Resources Laboratory.	7
1.5.3 Massachusetts Institute of Technology.	7
1.5.4 NASA.	7
1.5.5 University of North Carolina at Chapel Hill.	8
1.5.6 Commercial Products.	8
1.6 Summary	8
1.7 Thesis Overview	9

	Page
II. Software Library	10
2.1 Introduction	10
2.2 Input Device Software	10
2.2.1 Requirements.	10
2.2.2 Implementation.	11
2.2.3 Remote Input.	22
2.3 Graphics Display Software	25
2.3.1 Requirements.	25
2.3.2 Virtual Environment Objects.	26
2.3.3 Miscellaneous.	29
2.4 Summary	32
III. Head Mounted Display	33
3.1 Introduction	33
3.2 HMD-I	33
3.2.1 Description.	33
3.2.2 Problems.	33
3.3 HMD-II	35
3.3.1 Requirements.	35
3.3.2 Design.	35
3.3.3 Components.	38
3.3.4 Construction.	40
3.4 Summary	43
IV. Summary	46
4.1 Results	46
4.1.1 Software.	46
4.1.2 Head-Mounted Display.	47

	Page
4.2 Recommendations	47
4.2.1 Software.	47
4.2.2 Head-Mounted Display.	48
4.3 Future Applications	49
Appendix A. Manual Pages	50
Appendix B. HMD-II User's Guide	74
Appendix C. AFIT Geometry File Format	79
Bibliography	81
Vita	83

List of Figures

Figure	Page
1. VEDS Input Routines	12
2. CH Products Microstick Joystick	13
3. Joystick Data Structure	14
4. CIS Dimension Six Force-Torque Ball	15
5. Dimension Six Data Structure	16
6. VPL Dataglove	17
7. Dimension Six Data Structure	18
8. Polhemus 3-Space Tracker	19
9. Polhemus Initialization Code	20
10. Polhemus Field Orientation	21
11. Polhemus Data Structure	21
12. Server Main Task Pseudocode	23
13. Remote Input Routines	24
14. Virtual Environment Object	27
15. Filled Object Movement	29
16. Simulated Hand Model	30
17. HMD-I	34
18. Fresnel Press-on Prism Use in HMD-I	36
19. Side View of the First HMD-II Design	37
20. Top View of the Second HMD-II Design	38
21. Final HMD-II Design	39
22. LEEP Optics	40
23. Pin-cushion Distortion of LEEP Optics	41
24. Sony FDL-330 Color LCD Monitor	42

Figure		Page
25.	HMD-II	44
26.	HMD-II In Use	45
27.	Block Diagram of the HMD-II System	75
28.	10-Pin and 4-Pin Connector Pin Numbers	78

List of Tables

Table	Page
1. Joystick Byte Stream	14
2. Dimension Six Byte Stream	15
3. Polhemus Byte Stream	19
4. Server Configuration File Parameters	24
5. Client Data Retrieval Commands	25
6. HMD-II Wiring Harness Pinouts	77
7. AFIT Geometry File Format	80

Abstract

The design and development of a Virtual Environment Display System is presented. The system is composed of two main parts, a software library to support the development of virtual environment applications and a head-mounted display for viewing the virtual environment.

The software library provides support for numerous input devices including a VPL DataGlove, Polhemus 3-Space Tracker, Dimension Six Force-Torque Ball, and a joystick. Graphical objects can be displayed in either wireframe or shaded mode. Three dimensional pop-up menus are provided.

The head-mounted display is a fully-enclosed viewing device built using off-the-shelf components. The displays are color LCD televisions and are viewed through wide angle optics. Head position and orientation are tracked using a Polhemus 3-Space Tracker.

A 3-D VIRTUAL ENVIRONMENT DISPLAY SYSTEM

I. Introduction

1.1 Background

In 1965, Ivan Sutherland conceived of the *Ultimate Display*, one which would incorporate sensory inputs of all kinds (22:507). He envisioned one day being able to sit in computer generated chair, converse with a computer generated person, and the computer generated apple pies would smell just like Mom's. As noted by Chung *et al*, the "holodeck" from the television series "Star Trek—The Next Generation" is the final evolution of Dr. Sutherland's *Ultimate Display*, what we today call a virtual environment. (5:1).

The Armstrong Aeromedical Research Laboratory at Wright-Patterson AFB, OH, like Dr. Sutherland, is also in search of the ultimate display. Their ultimate display will come in the form of the Super Cockpit, a futuristic aircraft cockpit designed to enhance pilot performance through the use of innovative man-machine interfaces. One of those interfaces, and the focus of this thesis, is the Head Mounted Display (HMD). An HMD is a display device worn by the user in which the displayed image changes in response to the wearer's head motion.

1.2 Problem

At the Air Force Institute of Technology (AFIT) in 1988, Captain Bob Rebo constructed a virtual environment system based on an HMD of his own design (15). The system was constructed of low-cost, off-the-shelf components, and functioned remarkably well. The development of the first AFIT HMD system provided an

excellent testbed for the study of virtual environment applications. Captain Gary Lorimor has used the HMD-I system to view 3-D time-dependent data of an Air Force training exercise, and the system has been used to view simulated ballistic missile trajectories (12). However, being only a first generation system, there was insufficient time to develop a complete software library to allow the HMD-I system to be used to its full potential. In addition, being the first head-mounted display developed at AFIT, the system suffered from the following problems:

1. Poor display quality.
2. Excessive system weight.
3. Inability to fit every user.

1.3 Scope

The goal for this thesis is to develop a 3-D Virtual Environment Display System (VEDS). This system will consist of a library of software routines specifically tailored for use in a virtual environment system, and a head-mounted display for placing the user in the virtual environment. The software library will support many 2-D and 3-D input devices including a joystick, a Dimension 6 force-torque ball, a VPL Dataglove, and a Polhemus 3-Space Tracker. In addition, the library will support both 2-D and 3-D display devices. The HMD will be a stereo capable, enclosed color system with wide-angle optics. Off-the-shelf components will be used in the construction of the head-mounted display.

1.4 Assumptions

This thesis will require significant computer processing power to handle both the input devices and the graphics display. The AFIT computer graphics laboratory has three computers capable of performing the required tasks.

The first machine is a Silicon Graphics Inc. IRIS 3130 graphics workstation. The IRIS 3130 is a single processor system based on the Motorola 68020 capable of approximately two million operations per second (MIPS). It has a 19 inch color monitor and a 24 bit frame buffer. The IRIS 3130 is equipped with a proprietary *Geometry Engine* which greatly speeds graphics operations. However, the graphics operations supported by the *Geometry Engine* are limited to vectors and flat-shaded filled polygons; texture mapping is not supported (18). For serial input/output (I/O) the IRIS 3130 has three RS-232 ports.

The second machine that is available is a Sun Microsystems Inc. Sun-4/260 equipped with a TAAC-1 general purpose bit-sliced processor. The Sun-4/260 is a single processor system based on Sun's proprietary Symmetric Processor ARChitecture (SPARC) processor and is capable of eight MIPS performance. This machine also has a 19 inch color monitor but only an eight bit frame buffer. Serial I/O on the Sun-4/260 is limited to two RS-232 ports. The Sun-4/260 is a poor performer at high speed interactive graphics, but when paired with the TAAC-1 processor, its vector graphics performance greatly improves. The filled polygon performance of the TAAC-1 processor cannot match that of the IRIS 3130 and its *Geometry Engine* (21).

The third machine is a Digital Equipment Corporation (DEC) MicroVax 3. The MicroVax uses a DEC proprietary central processor operating at approximately three MIPS. The MicroVax has no graphics capability but has been equipped with twelve serial ports, more than enough to handle the several input devices required in this thesis.

A fourth type of machine, a DEC MicroVax 3/GPX was also available for use in this effort but had previously been found incapable of hosting virtual environment applications (12:25).

1.4.1 Input Server Hardware Selection. The basic requirements for the input server machine were that it (a) have enough serial ports to handle all the supported devices, (b) be powerful enough to handle input from all devices simultaneously, and (c) be connected to the other graphics lab machines through the lab network.

Only the MicroVax 3 met all these requirements. Though all machines in the AFIT graphics lab are connected to the lab network, and all have sufficient processing power to handle the input devices, the lack of serial ports ruled out all but the MicroVax 3.

1.4.2 Graphics Hardware Selection. The Silicon Graphics IRIS and the Sun-4 were each evaluated for its ability to perform the types of operations necessary in a virtual environment application. The following provides a summary of the evaluation criteria used in selecting a particular machine.

Graphics Capability: The first and most important criterion for selecting one machine over another was the interactive graphics capability. The evaluation of graphics capability included (a) the speed at which the machine could draw both vectors and filled polygons, and (b) the geometric primitives supported by the graphics hardware (if any) and by the graphics software.

Ease of Use: This criterion was used to judge how easy it was to write a simple graphics application. This related directly to the graphics software library support provided with the machine.

Code Development Tools: Since all of the candidate machines ran the Unix operating system, a strong code development environment existed on each. Of main concern was the programming language support since C++ was the desired implementation language. If C++ was not available then the C programming language would be used.

NTSC Output: This item was a necessity if the machine was to drive the head-mounted display, which was a fundamental assumption of this thesis.

The Silicon Graphics IRIS 3130 and the Sun 4/260 with TAAC-1 Application Accelerator were evaluated using the criteria above, and the following results were obtained.

Sun 4/260 with TAAC-1: The Sun-4 alone was incapable of supporting a virtual environment application. However, when used in conjunction with the TAAC-1 Application Accelerator, a sufficient level of graphics performance was possible, but only using wireframe models. The TAAC-1 has its own graphics library and can support NTSC output. The Sun-4 has support for C++, but TAAC-1 programs must be written in C.

IRIS 3130: The IRIS 3130 is a high performance graphics workstation specifically designed for 3-D and animation. It has an adequate code development environment, but does not support C++. The IRIS has dedicated graphics hardware making it very fast for graphics. NTSC output is supported. The graphics library is extremely easy to use, and provides all the functionality necessary for development of a virtual environment application.

Based on the selection criteria, the Silicon Graphics IRIS 3130 was clearly the only machine capable of supporting a virtual environment application.

1.4.3 Programming Language Selection. The choice of machine drove the choice of programming language for software development in this thesis. The IRIS 3130 supported numerous programming languages including FORTRAN, Pascal and C. However, C++ was not available despite several attempts by the author to obtain it. Therefore, C was chosen as the development language.

1.5 Literature Review

Virtual environment display systems have been around for almost twenty-five years, and the basic system has changed very little. Almost all virtual environment

systems consist of a head-mounted display with a sensor to track head movement and some basic software to update the view accordingly.

One fundamental difference in virtual environment systems is the type of HMD that is used, either see-through or fully enclosed. The see-through design superimposes the computer generated image on the real environment, while the fully enclosed approach allows the user to see only the computer generated images. Each type of system has advantages and disadvantages and finds application in different fields of study.

The following paragraphs present a somewhat chronological review of virtual environment display systems developed over the past twenty years. The list is by no means complete, but does describe the "major" systems that have been developed.

1.5.1 University of Utah. As early as 1966, Ivan Sutherland was experimenting with three dimensional displays. While working at MIT's Lincoln Laboratory, Dr. Sutherland developed one of the first, if not the first, head-mounted display. The optical system was primitive and only presented an image to one eye. However, in conjunction with an ultrasonic position sensor to track head movement, he was able to create the illusion of three dimensions (23:763). As he wrote at the time:

Even with this relatively crude system, the three dimensional illusion was real. Users naturally moved to positions appropriate for the particular views they desired. (23:763)

After moving to the University of Utah in 1968, Sutherland began work on a new head-mounted display. This system used two miniature cathode ray tubes (CRTs) in conjunction with some magnifying optics to present a virtual image eighteen inches in front of the eyes. The system was a see-through design, allowing the user to look out into the real world. With two separate CRTs, the system was capable of producing a stereo image. In fact, Sutherland was surprised at the favorable

response to the use of stereo. The user's head position was tracked using either of two methods, mechanical or ultrasonic (23:758-763).

1.5.2 Human Resources Laboratory. In 1981, Dennis Breglia and others from the Air Force Human Resources Laboratory at Williams Air Force Base in Arizona built a working head-mounted display. Their system used a small projector lens on top of a helmet worn by the user to project images on a dome screen (1:246). As the user turned his head, the projector moved accordingly, keeping what he should see projected directly in front of him. Since the image from the head-mounted projector only filled a portion of the user's field of view, separate stationary projectors were used to fill the rest of the dome. Because the system required a large dome to view the projected images, the system's use was limited.

1.5.3 Massachusetts Institute of Technology. Mark Callahan, in 1983, for his Master's thesis at MIT's Architectural Machine Group built an HMD using two Sony black and white CRTs mounted on a bicycle helmet (3). The two inch diagonal measure CRTs were positioned near the wearer's forehead with the display screens facing downward. The wearer looked through a pair of beam splitters, viewing the real and the virtual environment at the same time. Callahan used a Polhemus 3-Space Tracker to track head motion. One of Callahan's innovations was the use of interactive video disc for image generation.

1.5.4 NASA. Recent work at NASA's Ames Research Center has focused on the use of a head-mounted display to "facilitate natural interaction with complex operational tasks and to augment operator situational awareness" (7:77). NASA's main line of research with this system has been in remote robotic control and robotic telepresence, specifically geared toward space applications.

The NASA Ames' head-mounted display is a simple, low-cost design which uses wide angle optics and monochrome liquid crystal display (LCD) screens mounted on

a motorcycle helmet. Head motions are detected through the use of six degree-of-freedom tracking device mounted on the helmet (7:78).

1.5.5 University of North Carolina at Chapel Hill. Researchers at the University of North Carolina at Chapel Hill have built no less than fourteen virtual environment systems (2:3). Two systems of particular interest are Molecular Modeling Studies and the Virtual Building Walkthrough. Both of these systems were originally designed for use without an HMD; they used a large screen video projector to display the desired scene. Only recently have the researchers begun to use a head-mounted display with these systems (5:6).

1.5.6 Commercial Products. While all the previously described virtual environment display systems were developed mainly for research applications, several companies are beginning to produce commercial products using virtual environments. One such company is VPL Research of Redwood City, California. VPL's current product, RB2 (Reality Built for Two), has a stereoscopic head-mounted display for each user. The displays are constructed using color LCD displays and wide-angle optics, all mounted inside the frame of a scuba diving mask. The computer generated imagery is produced using four Silicon Graphics workstations, two for each user, one for each eye (14:17,22).

Another system currently under development is the Cyberspace system from Autodesk, Inc. The Cyberspace system uses the same head-mounted displays as the VPL RB2 system, but the display scenes are generated on an 80386-based PC equipped with special graphics accelerators. The cost of the Cyberspace system is one tenth that of the VPL system (14:22).

1.6 Summary

Virtual environment display systems are not new. Dr. Ivan Sutherland was experimenting with three dimensional displays as early as 1968. From his pioneering

work to today's commercial products, very little has changed. Advances in technology have made head-mounted displays lighter, easier to use, and now provide color displays. Head tracking has become simplified through the use of the Polhemus tracker. The computers driving the displays are more powerful and can draw more complex scenes. Dr. Sutherland's *Ultimate Display* has yet to be created, and this thesis effort will not create it either. This research will be limited to the development of a head-mounted display and a software library to support virtual environment applications.

1.7 Thesis Overview

This document is divided into four chapters. Chapter 1 has included a brief introduction, review of current literature, and statement of the problem. Chapter 2 contains a description of the software library developed for this thesis. Chapter 3 describes the design and construction of the AFIT HMD-II. Finally, chapter 4 reports the results of this effort, recommendations for further research, and possible future applications.

There are three appendices to this document. Appendix A contains Unix-style manual pages for all the routines in the VEDS software library. Appendix B is a user's guide to the HMD-II. Appendix C describes the AFIT geometry file format.

II. Software Library

2.1 Introduction

The first of two major components of the VEDS is the software library used to write virtual environment applications. The VEDS software library is divided into two main sections, the input device handling routines and the graphics display routines.

The input device routines handle the communication with the numerous input devices supported by the VEDS. This communication involves initializing a device, reading data from a device, and sending commands to a device. The input routines are designed to operate in either a local or remote mode.

The graphics display routines allow the user to create their own virtual environment. Support is provided in the library for both wireframe and filled objects with hidden surfaces removed. A simple terrain model is used to provide a "floor" for the virtual environment. In addition to the objects and terrain, pop-up menus and a simulated hand model are provided for virtual environment interaction.

2.2 Input Device Software

2.2.1 Requirements. The following requirements were identified for the input device handling portion of the VEDS software library:

1. *Standard Interface.* A primary design goal for the input device handling portion of the VEDS software library was to keep the programmer's interface consistent across all devices. In addition, ease of use for the application programmer was a prime concern. Three different programming interfaces were tried before a satisfactory design emerged.
2. *Device Support.* There were several input devices that had to be supported by the VEDS software library. These devices included:

- (a) CH Products Microstick Joystick
- (b) CIS Dimension Six Force-Torque Ball (Spaceball)
- (c) Polhemus 3-Space Tracker
- (d) VPL Dataglove

3. *Remote Input.* Another requirement was that the input devices should be able to be connected to any machine on the AFIT graphics ethernet, and the data made available to any other machine. This requirement would serve two purposes. First, the input sampling could occur continuously, since the data retrieval would be independent of the graphics processing running on a separate machine. Second, running the input sampling on a separate machine would allow a predictive tracking algorithm such as the Kalman filter (15:18) to be run without degrading graphics performance. However, as a consequence of running remote input, such predictive tracking has not been found to be necessary.

2.2.2 *Implementation.* The following paragraphs describe the implementation of the the VEDS software library. The software was developed on a DEC MicroVax 3 in the AFIT graphics lab. Each input routine was designed to be portable across a wide range of machines. All input routines have been tested on a Sun-4/260 and a Silicon Graphics IRIS 3130, as well as the Vax where the routines were developed.

2.2.2.1 *Standard Interface.* The interface to the input devices was designed to make it as simple to use and as device independent as possible. The interface was also designed to mimic the Unix standard I/O interface. The primary operations required, as in Unix, are to open the device, to read from the device, and to close the device. Table 1 contains a list of the VEDS input routines.

void	VE.open_xxx (char *ttyport, int speed);
void	VE.init_xxx (device dependent options);
void	VE.read_xxx (xxxData *data);
void	VE.close_xxx ();
void	VE.write_xxx (char *buf, int len);
int	VE.read_raw_xxx (char *buf, int len);
void	VE.parse_xxx (xxxData *data, char *buf);

Figure 1. VEDS Input Routines

The VEDS software library works much like the Unix standard I/O library. The device is opened using a `VE_open_xxx` library call with `xxx` replaced with the desired device name. The device must then be initialized with the `VE_init_xxx` call. The `VE_init_xxx` function is the only one of the input routines which doesn't take the same parameters for all devices; each device has its own initialization parameters.

Once the device has been opened and initialized, subsequent calls to the routine `VE_read_xxx` will return the data from the device. The data is returned in a device dependent structure described in the following sections. When input is complete, the device should be closed using the `VE_close_xxx` function call.

Three additional routines are provided in the standard interface to the input devices. These routines will not normally be used, but can prove useful in certain circumstances. In fact, the preceding routines are built on top of these routines. The routine `VE_write_xxx` permits communication with those devices which will accept host commands. `VE_read_raw_xxx` reads the raw byte stream coming from the device. The routine `VE_parse_xxx` separates the raw byte stream from the device into the device dependent structure.

2.2.2.2 Device Support. Four separate input devices are supported. They are a CH Products Microstick Joystick, a CIS Dimension Six Force-Torque Ball, a VPL DataGlove, and a Polhemus 3-Space Tracker. The methods by which each of

these devices is supported are explained in the following sections.

Joystick. The Microstick Joystick from CH Products (shown in Figure 2) is a 2-D input device with a 4096×4096 pointing resolution and three buttons (4). The joystick interfaces to the host computer system through a standard RS-232 connection with an external +5V power source. Baud rates from 300 to 19200 are supported, and the joystick can be configured from the host or by DIP switches located on the bottom of the unit. Output from the joystick is in an ASCII byte stream as shown in Table 1.

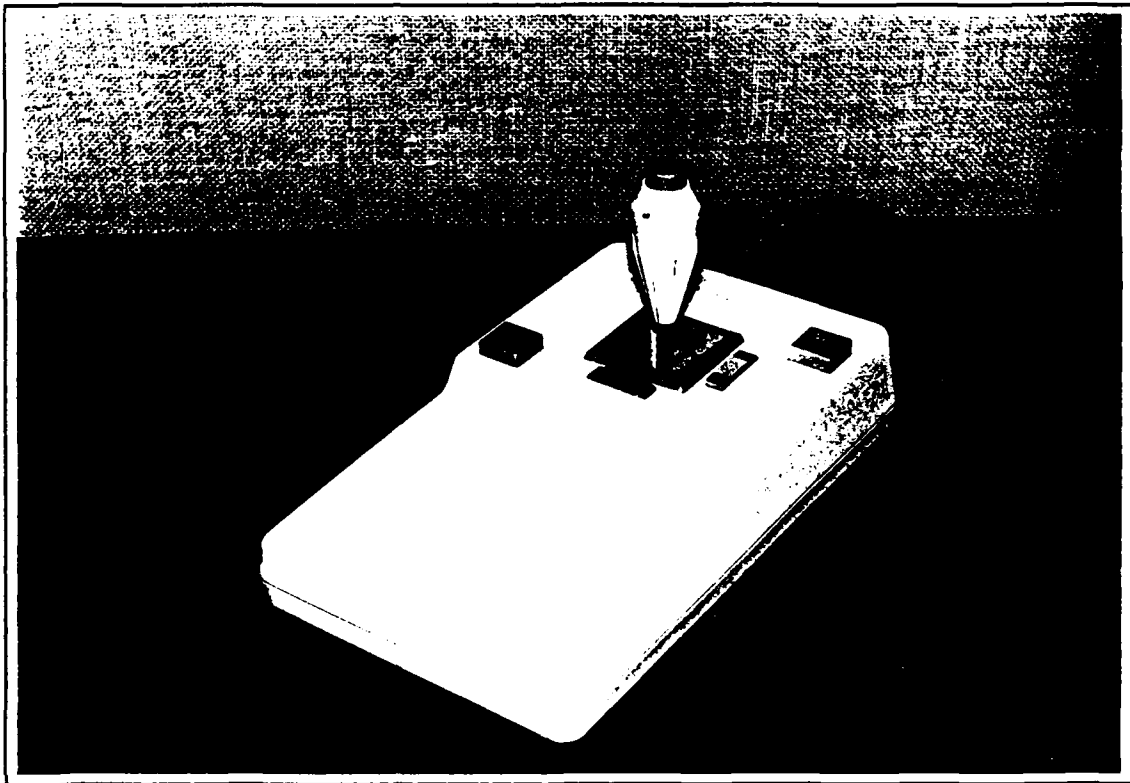


Figure 2. CH Products Microstick Joystick

This ASCII byte stream is converted through the `VE_parse_joystick` routine into the data structure shown in Figure 3 and defined in the file `joystick.h`.

Table 1. Joystick Byte Stream

Delim	Button1	Delim	Button2	Delim	Button3	Delim	X Value	Delim	Y Value	Delim
\$	0/1	,	0/1	,	0/1	,	X	,	Y	CR/LF

```
typedef struct
{
    short button;
    short x;
    short y;
} JoystickData;
```

Figure 3. Joystick Data Structure

CIS Dimension Six. The CIS Dimension Six force-torque ball (shown in Figure 4) is a six degree of freedom input device combining the functions of a joystick, a button box, and a dial box (6). The ball can be rotated about the three primary axes and translated in three directions. Force sensors inside the ball register the amount of force and torque being applied to the ball and send this information to the host over a standard RS-232 interface. Eight buttons are provided. The Dimension Six can operate in either an ASCII or binary mode at several baud rates. The VEDS software only supports the ASCII mode of operation for the Dimension Six. The device can be configured from the host or through DIP switches located on the bottom of the unit.

The values from the Dimension Six depend on the data format selected. Using the SB_FORCE format, translation and rotation values range from -127 to 128 and a button press returns the button number three times positive, then three times negative. In the SB_VOLTAGE format, translation and rotation values range from -999 to 999 and a button press changes the appropriate 0 to 1 in the string 00000000. Output from the Dimension Six is in the format shown in Table 2.

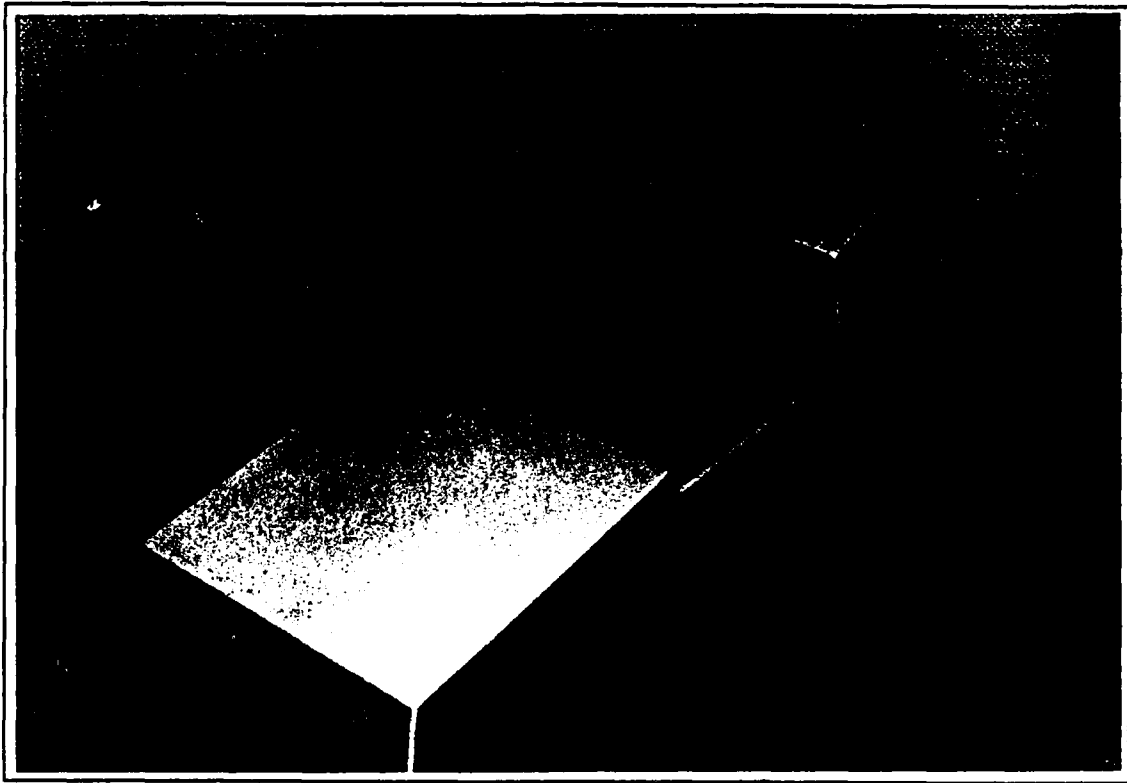


Figure 4. CIS Dimension Six Force-Torque Ball

The ASCII byte stream from the Dimension Six is converted by the routine `VE_parse_spaceball` into the data structure shown in Figure 5 and defined in `spaceball.h`.

Dataglove. The Dataglove (shown in Figure 6), manufactured by VPL Research, Inc. of Redwood City, CA, is a "computer input device which converts hand gestures and positions into computer-readable form" (27). The Dataglove

Table 2. Dimension Six Byte Stream

Mode	Xtrans	Ytrans	Xtrans	Xrot	Yrot	Zrot	Button
FORCE	± 128	± 128	± 128	± 128	± 128	± 128	± 8
VOLTAGE	± 999	± 999	± 999	± 999	± 999	± 999	0/1 for each button

```
typedef struct
{
    short xtrans;
    short ytrans;
    short ztrans;
    short xrot;
    short yrot;
    short xrot;
    short button;
} SpaceballData;
```

Figure 5. Dimension Six Data Structure

system consists of a lycra glove fitted with fiber optic cables, connected to a separate control unit. Through the fiber optic cables, the control unit can sense flexion and extension of the fingers, and can sense hand position through the use of a built-in Polhemus tracker.

The Dataglove control unit is connected to a host system through either an RS-232 or RS-422 cable. The unit can operate at several baud rates from 300 to 19200. There is a separate RS-232 "user port" through which the Dataglove control unit can manipulate another device. All port settings are accomplished with DIP switches.

The Dataglove is the only one of the supported input devices that does not send ASCII data records, only binary records are transmitted. Using binary data precludes the use of a unique delimiter in the output record since the delimiter may appear in the data. Several unsuccessful attempts were made to write code that would read the Dataglove correctly. Eventually, code was obtained from Simgraphics, Inc., which properly read the Dataglove output records. The VEDS software interface to the Dataglove makes calls to the Simgraphics Dataglove library.

Because the Simgraphics library is used for manipulation of the Dataglove,

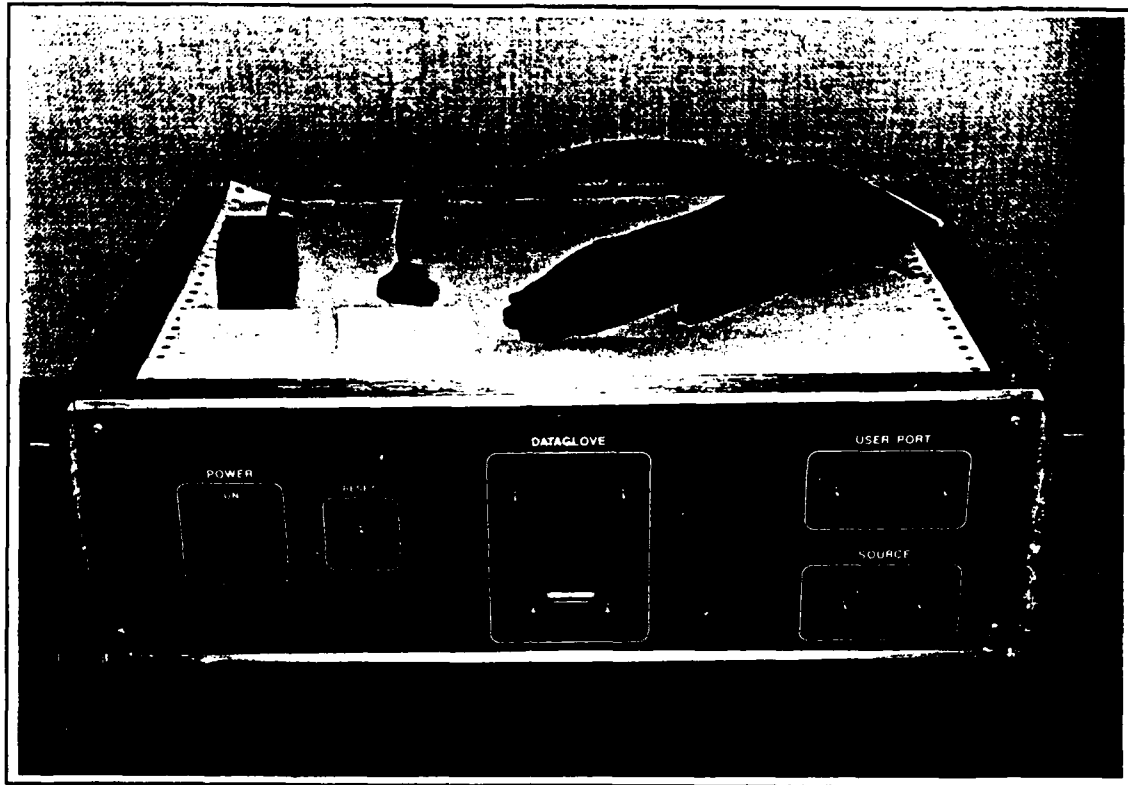


Figure 6. VPL Dataglove

some VEDS routines for dealing with the Dataglove are inoperative. These routines are `VE_init_dataglove`, `VE_write_dataglove`, and `VE_read_raw_dataglove`. This loss of functionality is due to limitations in the Simgraphics library.

The `VE_read_dataglove` routine, however, works as expected and returns the structure shown in Figure 7 and defined in `dataglove.h`.

Polhemus. The Polhemus 3-Space tracker (shown in Figure 8) is a six degree-of-freedom input device that uses a low frequency magnetic field to determine the position and orientation of a sensor in relation to a source or other specified frame of reference (13:1-1). The Polhemus system consists of three parts: a system electronics unit (SEU), one or two sources, and one to four sensors. The source is fixed at a position near the center of the area of intended use. The area

```
typedef struct
{
    Position pos;
    Orientation or;
    short flex[NUMSENSORS];
    Byte gesture;
    char gesture_name[16];
} DatagloveData;
```

Figure 7. Dimension Six Data Structure

of use should be free of all metallic objects, since these warp the magnetic field produced by the source and cause false readings (13:2-13).

The Polhemus connects to the host computer system through a standard RS-232 connection. Baud rates from 300 to 19200 are supported; configuration is by DIP switches on the back of the unit. Both ASCII and binary communication modes are supported, however the VEDS software only supports the ASCII mode.

Numerous data items can be obtained from the Polhemus while it is in operation. These items include cartesian coordinates, orientation angles, directional cosines for *X*, *Y*, and *Z*, and quaternions. However, not all these data items need be reported. The Polhemus is completely programmable from the host and the user has complete control over the data items that are reported.

The code in Figure 9 is an excerpt from the VEDS routine `VE_init_polhemus`. First, the Polhemus is cold reset. This ensures that all settings have been returned to their default condition. Next the Polhemus boresight is reset to default, increment is set to zero, ASCII format is selected, and the unit of measure is set to inches. The operational envelope is then set to the maximum sixty five inches in all directions. Next the operational hemisphere is set to have its zenith on the *Z* axis as shown in Figure 10. Finally the output record is set to that shown in Table 3. The commands

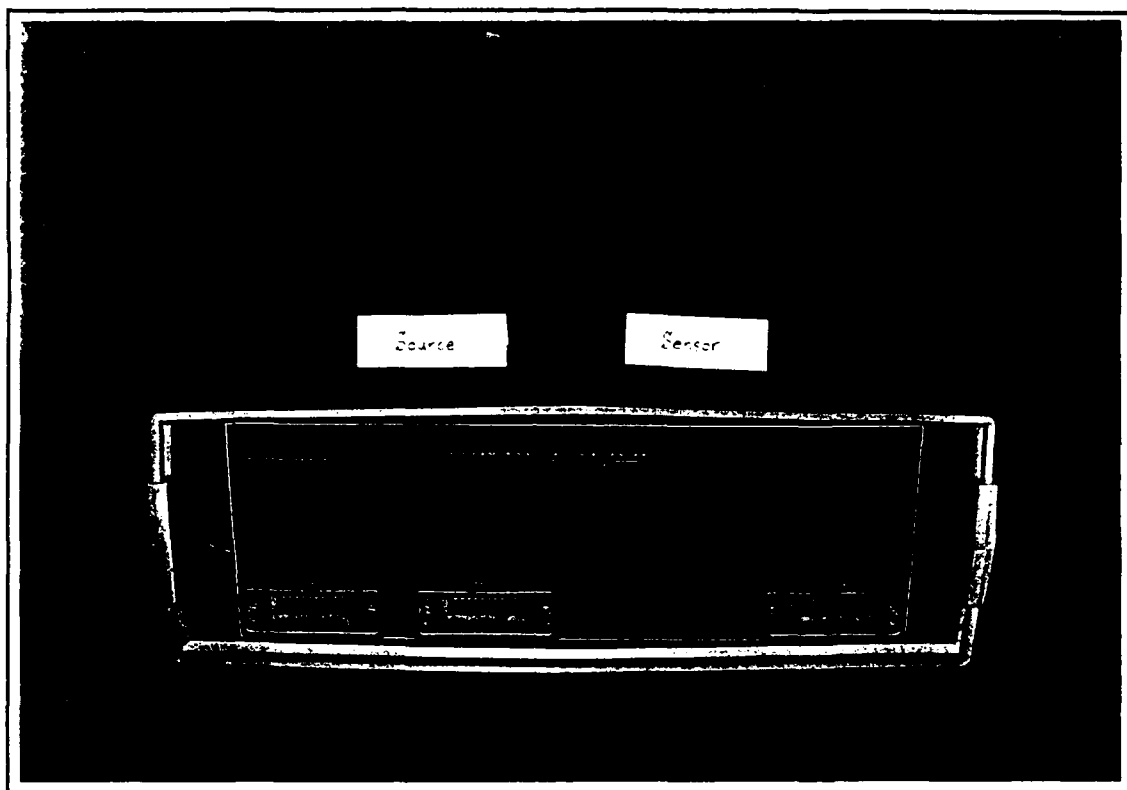


Figure 8. Polhemus 3-Space Tracker

sent to the Polhemus are fully explained in (13).

Table 3. Polhemus Byte Stream

Position			Orientation			X-axis cosines			Y-axis cosines			Z-axis cosines			Delim	
X	Y	Z	Asimuth	Elevation	Roll	X	Y	Z	X	Y	Z	X	Y	Z	CR	LF

This data record from the Polhemus is parsed by the `VE_parse_polhemus` routine into the data structure shown in Figure 11 and defined in `polhemus.h`.

The Polhemus can operate in either a polled or continuous mode. Using polled mode requires that the timing between the poll and the receipt of poll be precise. If a read of the data takes place before data has been received, the read will return invalid data. If the read takes place too late, time will be wasted that could have been spent

```

buf[0] = '^Y';                                /* Cold reset */
VE_write_polhemus(buf, 1);
sleep(6);

strcpy(buf, "b1\rI0\rFcU");                    /* Reset boresight,
VE_write_polhemus(buf, strlen(buf));            Increment to 0,
sleep(1);                                       ASCII format,
                                              unit inches */

strcpy(buf, "V1,65,65,65,-65,-65,0\r"); /* Define envelope */
VE_write_polhemus(buf, strlen(buf));
sleep(1);

strcpy(buf, "H1,0,0,1\r");                    /* Define hemisphere */
VE_write_polhemus(buf, strlen(buf));
sleep(1);

/* Set output record format to the following: */
/* 2 = Cartesian Coordinates (X, Y, Z) */
/* 4 = Orientation Angles (azimuth, elevation, roll) */
/* 5 = X-axis direction cosines (x, y, z) */
/* 6 = Y-axis direction cosines (x, y, z) */
/* 7 = Z-axis direction cosines (x, y, z) */
/* 1 = CR, LF */

strcpy(buf, "02,0,4,0,5,0,6,0,7,1\r");
VE_write_polhemus(buf, strlen(buf));
sleep(1);

```

Figure 9. Polhemus Initialization Code

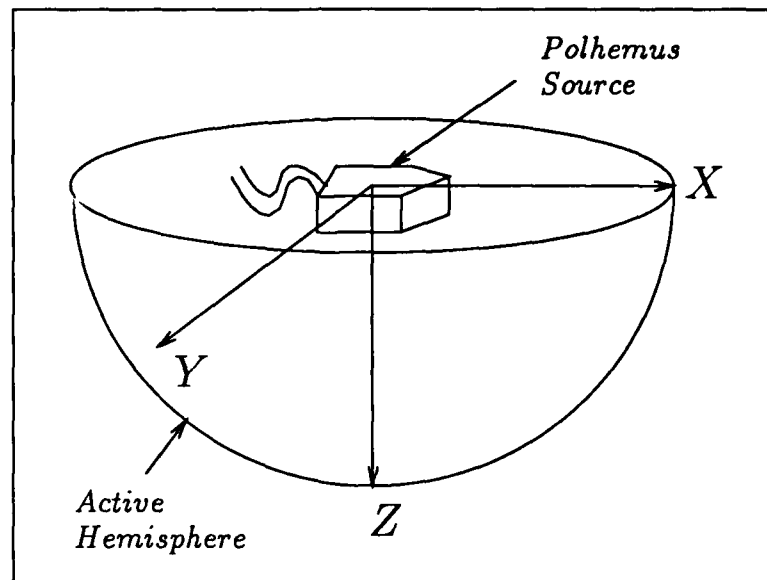


Figure 10. Polhemus Field Orientation

```
typedef struct
{
    Position pos;
    Orientation or;
    DirectionCosine xdc;
    DirectionCosine ydc;
    DirectionCosine zdc;
} PolhemusData;
```

Figure 11. Polhemus Data Structure

doing something else. The solution to the problem is to use continuous mode. The `VE_init_polhemus` routine switches the Polhemus into continuous output mode.

2.2.3 Remote Input. The ability to have the input devices physically attached to a machine other than the one driving the graphics display was a fundamental requirement for the VEDS input routines. By devoting an entire machine to sample the input devices, it was possible to place all the input devices into a continuous output mode and still be able to sample fast enough. In addition, this "input machine" could support many devices at one time, while a machine having to do the graphics processing at the same time might become overloaded.

The remote input code was developed using the client/server model as described in (11:17). The machine with all the input devices attached would act as the server, while the machine handling the graphics processing would be the client. To implement such a model, three facilities were needed. The first was a communications medium through which the client and server could exchange information. The second was a server program that would continuously sample the input devices and watch for data requests from the client processes. The third was a set of user callable client routines which would retrieve data from the server.

All machines in the AFIT graphics lab are attached to the lab's ethernet local area network. This network provided the perfect communications medium for the implementation of the remote input facility. The network, operating at ten megabits per second, provided more than enough bandwidth for the small data records being transmitted.

The server and client portions of the VEDS were implemented using the Berkeley interprocess communication (IPC) services in the Unix operating system. The reader is referred to (11:17) for an in-depth discussion of the Unix IPC facilities.

2.2.3.1 *Server*. The server is divided into three main parts: (a) the server main program, (b) the device reader, and (c) the network reader.

Referring to Figure 12, when the server starts up, the configuration file is read in. If no errors are found in the configuration file, a shared memory segment (for subprocess communication) is opened. Then, for each attached device, a new *device reader* process is started which opens the device (if possible) and begins sampling data. As the data is read from the input devices, it is written into the shared memory segment. Once the *device reader* tasks have been spawned, the main server waits for connection requests from client programs. Once a connection request is received from a client, a *network reader* process is started which listens for data requests from the client. Upon receipt of a data request, the *network reader* process reads the current device data record from shared memory and transmits it to the client. After spawning the *network reader* process, the main server process returns to waiting for client connection requests. Thus, to serve on client program using one input device, three server processes are required, the main server process, the *device reader* process, and the *network reader* process. The current implementation limits the number of clients to five and the number of input devices to four (only one of each supported input device).

```
read configuration file
open shared memory segment
spawn appropriate device reader tasks
while (not done)
    wait for client connection
    spawn network server task
```

Figure 12. Server Main Task Pseudocode

The server program uses a configuration file to specify which input devices are to be used. This file has the format shown in Table 4. As an example, specifying

Table 4. Server Configuration File Parameters

Keyword	Path	Baud Rate
joystick	/dev/ttyXX	300 – 19200
spaceball	/dev/ttyXX	300 – 19200
polhemus	/dev/ttyXX	300 – 19200
dataglove	/dev/ttyXX	300 – 19200
calibration_file	path/filename	N/A
gesture_file	path/filename	N/A

“polhemus /dev/tty02 19200” indicates that the Polhemus tracker is connected to /dev/tty02 running at 19200 baud.

The keywords `calibration_file` and `gesture_file` are specific to the Dataglove and should be specified whenever the Dataglove is used, unless the defaults are acceptable. The `calibration_file` should be the name of a hand calibration file generated with the Simgraphics Dataglove Test and Calibration program. This allows the Dataglove to be tailored to each user’s hand. The `gesture_file` should be the name of a hand gesture file generated with the Simgraphics Gesture Editor.

2.2.3.2 Client. The VEDS remote input routines available to client programs were designed to parallel the input routines for each of the input devices. The semantics of the remote input routines remain the same as those for using the input devices directly. Figure 13 shows the remote input routines available to client programs.

```
void    VE_open_remote(String host);
void    VE_read_remote(Byte command, xxxData *data);
void    VE_read_raw_remote(Byte command, Byte *buf, int size);
void    VE_close_remote();
```

Figure 13. Remote Input Routines

Table 5. Client Data Retrieval Commands

Command	Action
SEND_DATAGLOVE	Retrieve Dataglove Data
SEND_JOYSTICK	Retrieve Joystick Data
SEND_POLHEMUS1	Retrieve Polhemus Data for Sensor 1
SEND_POLHEMUS2	Retrieve Polhemus Data for Sensor 2
SEND_SPACEBALL	Retrieve Spaceball Data

VE_open_remote is used to open a network connection to the host machine running the input server program as described in Section 2.2.3.1. VE_read_remote causes a request for data to be sent to the *network reader* task on the server machine. The data record that will be sent by the server is based on the command argument to VE_read_remote. The available commands are shown in Table 5. The VE_close_remote routine causes the server process to exit and closes down the network connection to the server host. As a result, when multiple clients are used, the first to call VE_close_remote will cause all server processes to exit, in effect disabling all clients.

2.3 Graphics Display Software

The second major portion of the VEDS software library is the graphics display routines. The graphics routines allow the user to build a virtual environment by creating objects which respond to the user's input.

2.3.1 Requirements. The requirements for the graphics display portion of the VEDS software library were simple given sufficient graphics capability, but proved much more difficult given the graphics hardware available for this project. The requirements were:

1. Provide a library of software routines to permit the construction and display of a virtual environment. The library should provide as a minimum the following

functions:

- (a) Build and display a wireframe representation of any arbitrary geometric object.
 - (b) Build and display a filled (with hidden surfaces removed) geometric object.
 - (c) Permit arbitrary translation, rotation, and scaling of both type of objects.
 - (d) Provide a pop-up menu system which can exist within the virtual environment.
2. Be highly functional and easy to use. Only a few library calls should be necessary to create a virtual environment.
 3. Provide an adequate update rate for the displayed image. Adequate will be defined as 10 frames per second.

2.3.2 Virtual Environment Objects. The VEDS graphics routines are based on the display of objects, which can be any convex polygon or convex collection of polygons. Each object within VEDS maintains a state which contains such items as its geometric description, color, current transformations, etc. This state information is contained in a `VE_Object` structure shown in Figure 14.

2.3.2.1 Object Hidden Surface Elimination. For filled objects, hidden surfaces must be removed for the displayed object to look *correct*. There are several methods for properly removing hidden surfaces from an object (24).

The VEDS library uses a technique known as the Binary Space Partitioning (BSP-tree) algorithm first introduced in (8) and later refined in (9). The BSP-tree works by exploiting the principle of separating planes. Thus, given a plane in a three dimensional scene, no polygon on the viewer's side of the plane can be obscured by a polygon on the other side of the plane. Using this simple notion, the BSP-tree algorithm builds a binary tree of polygons from the original polygon list. Once the

```

typedef struct
{
    Colorindex color;
    Boolean visible;
    Position position;
    SmallPosition rotation;
    SmallPosition scale;
    float distance;
    Position centroid;
    Position bb[BBOXSIZE];
    Displaytype type;
    Object wire;
    TreeNode *bsp;
    Matrix matrix;
    Matrix inverse_matrix;
} VE_Object;

```

Figure 14. Virtual Environment Object

tree has been constructed, image generation is a matter of traversing the tree in the correct order while painting each polygon encountered during the traversal (9:66).

2.3.2.2 Object-to-Object Visibility. One of the major drawbacks of the BSP-tree is that it is "limited to static world models (since whenever the world model changes, the preprocessing data restructuring step must be invoked)" (9:65).

For the VEDS, a modification to the use of the BSP-tree has been made. Instead of using all the objects in the virtual environment to generate the BSP-tree, each object has its own BSP-tree description. This eliminates having to re-generate the tree whenever an object moves, but does not preclude one object from intersecting another.

To solve the object-to-object visibility problem, another method was developed. The basic idea is to know the relationship of each object to the viewer. In other words, if the distance to each object from the viewer can be determined, then

the objects can be displayed in the correct order. The distance from the viewer to an object is calculated to the object's centroid. The objects are then sorted by distance and displayed in a farthest to nearest order.

Note that this method does not completely solve the object-to-object visibility problem, but is a close approximation. It is possible for objects to intersect even though they are correctly sorted by distance. However, testing has shown that in the interactive virtual environment, this approximation is not a problem.

2.3.2.3 Filled Object Movement. The BSP-tree algorithm was originally developed for the situation "where the world model changes less frequently than the viewpoint or direction of view of the observer" (9:65). However, in a virtual environment, this situation may not always exist; it would be nice to pick up a virtual object and move it freely, and still have the hidden surface problem solved.

The BSP-tree can still be used in this situation with slight modification. If an object moves, it is as though the viewer's eye were moved in the opposite direction. For instance, in Figure 15, if the cube shown in (a) is rotated 90° about the Z axis and the viewer doesn't move, then the green (G) face will now be visible as in (b). However, the same effect can be accomplished by moving the viewer -90° about the Z axis as shown in (c).

This principle is exploited in the VEDS graphics routines to allow arbitrary movement of objects described by a BSP-tree. Since a BSP-tree object must remain stationary, for the object to appear to move, the viewer must move in the opposite direction. This can be accomplished by transforming the viewer's position by the inverse of the object's transformation. The object's transformation can be represented by a 4×4 matrix (M), so the new viewer's position $[x \ y \ z \ 1]$ can be computed as shown in Equation 1.

$$[x \ y \ z \ 1] = [x' \ y' \ z' \ 1] [M]^{-1} \quad (1)$$

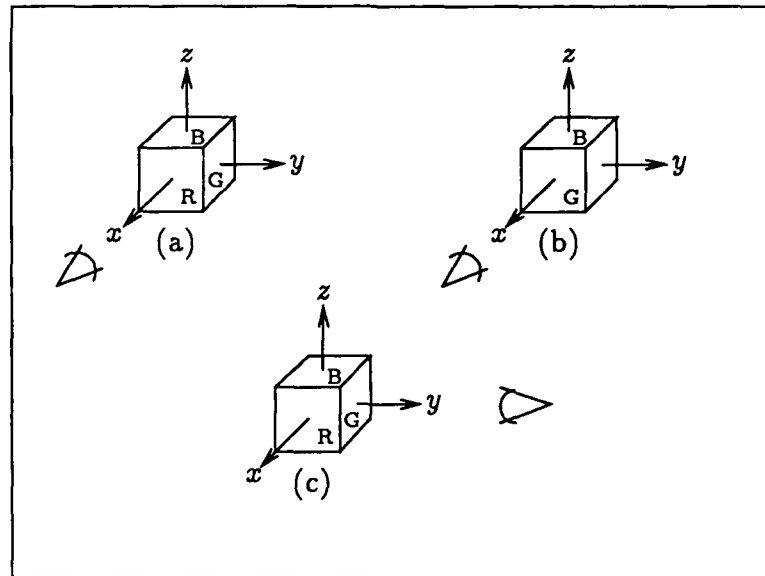


Figure 15. Filled Object Movement

2.3.2.4 Building Objects. Objects used in the virtual environment are built from a geometric description contained in an AFIT geometry file described in Appendix C. A call to the routine `VE_build_object` will build both a wireframe and filled description of the object.

Building a virtual environment object (`VE_Object`) from its geometric description is a two pass process. First, the geometry file is read in, and a linked list of polygons is built. As the list is being built, a Silicon Graphics object is created using the move and draw commands from the Silicon Graphics graphics library. Once the linked list has been built, the list is passed to the VEDS routine `_VE_build_tree` which creates the BSP-tree description of the object.

2.3.3 Miscellaneous. In addition to the virtual environment objects, three additional graphical entities are provided by the VEDS graphics routines. These include a simulated hand model, pop-up menus and information panels, and terrain.

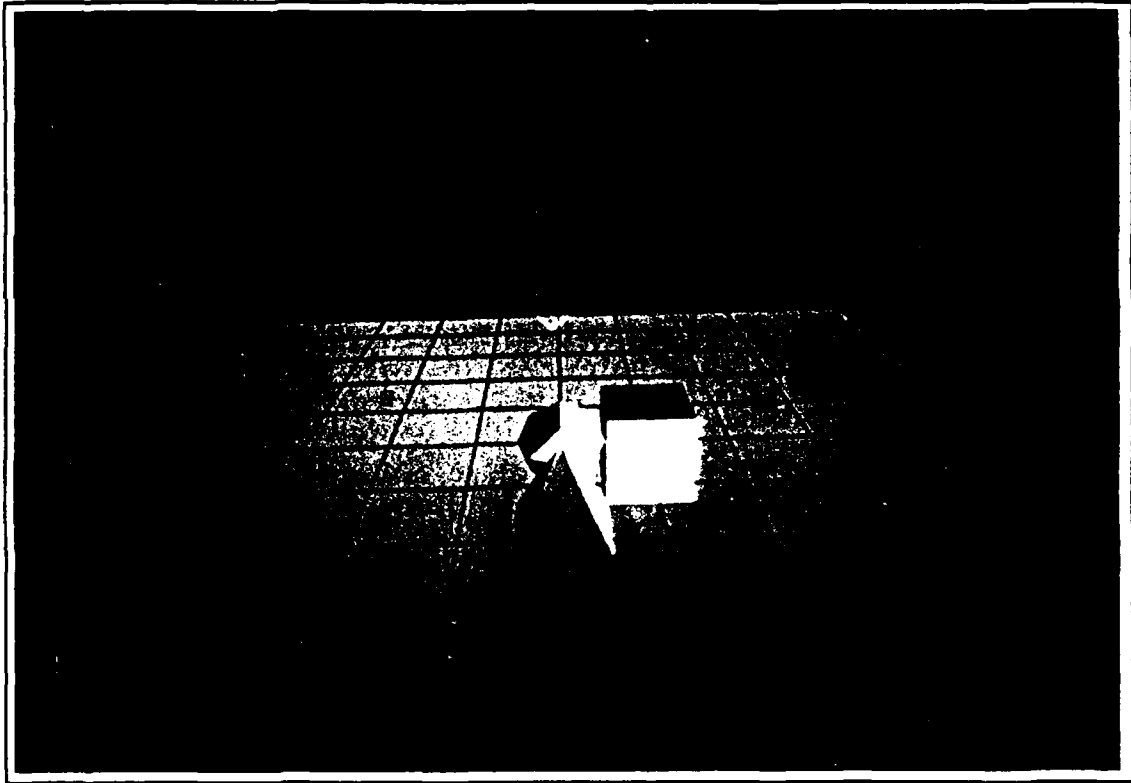


Figure 16. Simulated Hand Model

Each of these is described in the following sections.

2.3.3.1 Hand. When using the VPL DataGlove it is convenient to see the current position and orientation of the hand in the virtual environment. To do this, a simulated right hand is modeled using polygons and displayed in wireframe. The hand responds to position, orientation, and finger flex data from the DataGlove. An example of the hand in use in a virtual environment is shown in Figure 16.

2.3.3.2 Menus and Information Panels. The VEDS library provides pop-up menus and information panels. Menus provide for user choice selection, while information panels are used only for displaying textual information. These menus and panels are created within the virtual environment and may be manipulated just

like any other object in the environment. Menu items may be highlighted and menu picks made as the user desires.

Text within the menus and information panels is created by drawing up to six vectors for each character. This vector character set was designed on graph paper by the author. The per character vector count was held at six to maintain a reasonable update rate while at the same time providing a readable character set. The character set is limited to the uppercase letters, the numbers, and the space character. Lower case letters are mapped to uppercase. Undefined characters are mapped to the space character.

Menus exist for the duration of the program in which they are used, and may be used any number of times. Information panels exist only for as long as they are displayed. Subsequent requests for an information panel destroys the original and a new one is created.

2.3.3.3 Terrain. The display of terrain is a complex problem that has been the subject of much research (17). For this thesis, several methods for displaying terrain were investigated.

The first method investigated for displaying terrain was the use of a multiple level of detail terrain database. This involves having multiple terrain databases, only one of which is used at any one time. Database selection is based on the viewer's *altitude* above the terrain. A terrain model with more detail will be displayed when the viewer gets closer to the terrain. This method of terrain display was found unsuitable for this thesis because of the time required to create the terrain database and the need for special modeling tools (17).

Another method investigated was one similar to that used in the Fiber Optic Guided Missile (FOG-M) simulator (19). In FOG-M, the terrain is divided into grid squares. Each terrain square in the grid is treated as a separate object. These terrain objects are then sorted by distance from the viewer, and displayed in a back to front

order. Since a similar method is used in VEDS for object display, this seemed like a usable approach. However, one key question that must be answered is "how many grid squares should be used?" If the grid is too fine, the update rate will be slow. If the grid is not fine enough, objects will "disappear" behind grid squares as a result of the object sort. This method was also found unsuitable because of the specific nature of its application. The technique was not general enough for use in the VEDS.

The VEDS library provides only the most simplistic representation of terrain. This simplistic representation is one polygon of a user specified size and color with a 10×10 wireframe grid superimposed on the polygon. This polygon provides a "floor" for the virtual environment and should be sufficient for the envisioned applications of the VEDS.

2.4 Summary

The VEDS software library consists of two main parts, the input device routines and the graphics display routines. The routines for handling the input devices are designed for ease of use and consistency of interface. The input routines permit both local and remote operation of all the supported input devices. The local routines permit greater control over the device at the expense of the overhead of handling all the I/O. Remote access to a device frees the client from constantly reading the device and possible suffering a degradation in performance.

The graphics display routines are designed to permit easy construction of a virtual environment. There are routines for building and displaying virtual objects and terrain. Both wireframe and filled objects can be built and display with hidden surface properly removed.

III. Head Mounted Display

3.1 Introduction

A head-mounted display is a device worn by the user which presents an image to the eyes based on where the person is looking. The image the user sees could be either computer generated or live video, but the illusion of reality is created when the scene changes in response to the user's head movement.

The head-mounted display is the heart of the Virtual Environment Display System. Through the use of the head-mounted display, the VEDS user can become totally immersed inside the virtual world, making it that much more real.

The head-mounted display constructed for this thesis is the second generation of displays made at AFIT. The first, HMD-I, was constructed in 1988 by Captain Bob Rebo, and is briefly described in Section 3.2. The second generation display constructed for this thesis will be known as HMD-II.

The HMD-II is constructed from several major components: a set of high quality optics, miniature LCD color televisions, a scuba mask, and straps.

3.2 HMD-I

3.2.1 Description. For his master's thesis, Captain Bob Rebo designed and built the first generation of head-mounted displays at AFIT. Captain Rebo's HMD was built using a bicycle helmet for support, two small color LCD televisions for image display, and crude optics for display visibility. Head position was tracked using a Polhemus 3-Space tracker. See (15) for a complete description of HMD-I.

3.2.2 Problems. Though HMD-I functioned adequately, it suffered from several drawbacks. The first drawback was the weight of the system. The bicycle helmet and the frame used to hold the televisions, plus the weight of the televisions

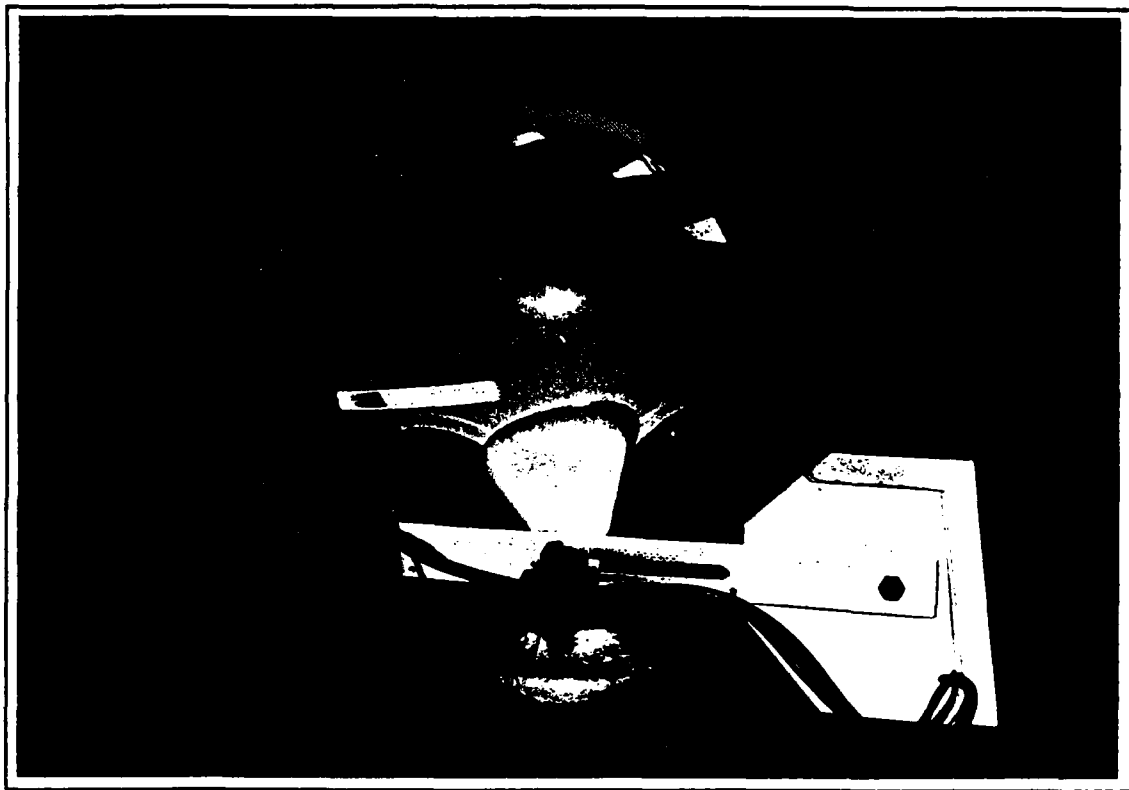


Figure 17. HMD-I

themselves made HMD-I uncomfortable to wear for more than a few minutes at a time. The second drawback was the optics which allowed the wearer to focus on the screens. Captain Rebo used two pair of eyeglasses, with +10 and +12 diopter lenses, to permit focus down to the small distance required. However, since the television screens he used were too big, Fresnel press-on prisms were required to see the screens which were positioned at angles to the eyes. The prisms blurred the displayed image considerably, causing eye strain and greatly reducing the effectiveness of the image. The third drawback was the one-size-fits-all limitation of the helmet; at least one prospective wearer was unable to use the system because the helmet was too small. See Figure 17 to better understand the design of HMD-I.

3.3 HMD-II

3.3.1 Requirements. Using Captain Rebo's HMD-I as a starting point and attempting to avoid the problems with the original head-mounted display, requirements were developed for the new HMD-II.

1. *Improve the image quality.* The image quality inside the HMD-II would never rival that of a computer monitor, or a television for that matter, but a display was needed that was pleasant to use for a prolonged length of time.
2. *Reduce the total system weight.* The HMD-II should be comfortable to wear for extended periods of time, at least thirty minutes.
3. *Make HMD-II more adjustable and easier to wear.* The display should be easy to don and remove, and should fit everyone.

3.3.2 Design. Using the three requirements as a basis for the design, work was begun to formulate several prototypes. Improving the image quality was the overriding concern in the development of HMD-II, as poor image quality was the foremost failing of HMD-I.

The main cause of the poor image quality in HMD-I was the Fresnel press-on prisms. The prisms were necessary because the LCD screens could not be aligned directly in front of the eyes (see Figure 18). The Fresnel prisms work well, are lightweight and easy to use, but cause a great deal of optical distortion. To improve the image quality, the use of press-on prisms would have to be changed.

Another cause of the poor image quality was the optics used in HMD-I. Captain Rebo used two pair of ordinary eyeglasses fitted with +10 and +12 diopter lenses. Two pair were required to get proper magnification. However, using two pair also distorted the visible image to some extent. To eliminate the distortion, a new set of optics would have to be used.

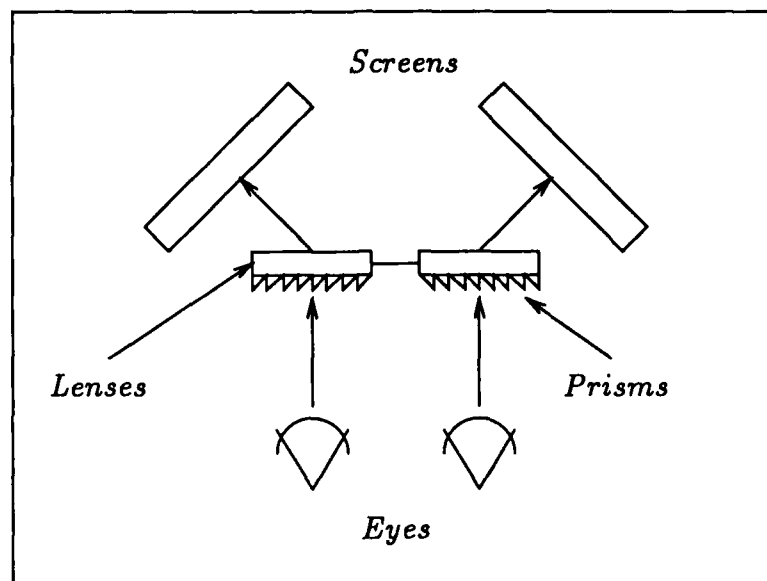


Figure 18. Fresnel Press-on Prism Use in HMD-I

Working solely to improve image quality, three initial designs were proposed. The first two were attempts to eliminate the optics from the system altogether (see Figures 19 and 20). By using a series of mirrors, the effective distance from the eye to the screens could be lengthened, and the optics eliminated. There were at least two major drawbacks to each of these designs. First, the requirement to reduce the system weight would not be met. The mirrors required by the new design, in combination with the full helmet, could be too heavy for extended use. Second, each design still used a helmet as the mounting platform, which would again limit its use to only those persons with the proper head size.

Another drawback to the designs shown in Figures 19 and 20 would be the narrow field of view (FOV) they would provide. Though narrow FOV displays have been used before, and with great success, it was preferred that HMD II, like HMD-I, would have a wide FOV (25). The wide FOV gives the wearer the illusion of being

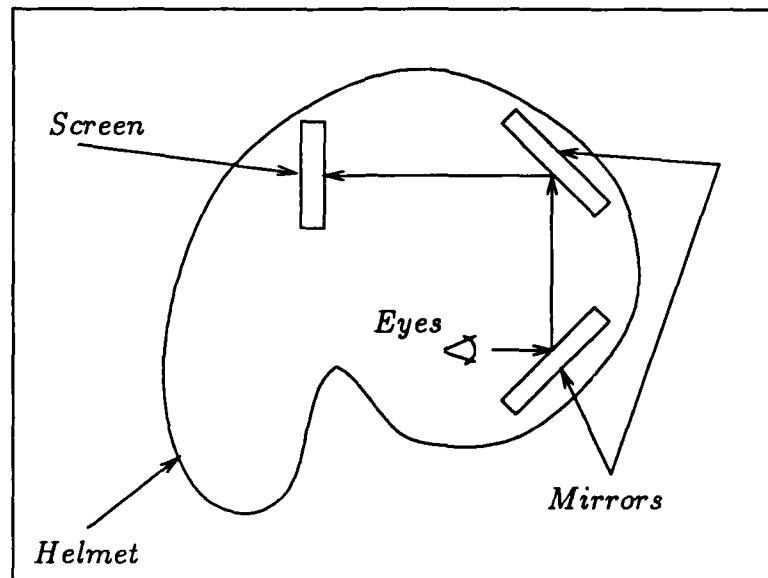


Figure 19. Side View of the First HMD-II Design

immersed in the virtual environment.

Since a wide FOV was desired, a different type of designed was called for. The optics could not be eliminated if the wide FOV was to be supported, so a new set of optics was required. Drawing on the virtual environment work done at NASA Ames, a set of optics designed by Eric Howlett and built by Pop-Optix Labs were chosen as the optics for HMD-II (7). See Section 3.3.3.1 for a complete description of the optics.

Using the LEEP optics, as they are called, satisfied the requirement for an improved image display. To satisfy the requirement to reduce the weight of the system, the helmet was removed from the design and replace with lightweight support straps. The design now resembled a pair of ski goggles as shown in Figure 21. Using this goggle-type design would permit greater adjustability than the helmet and allow more people to use it.

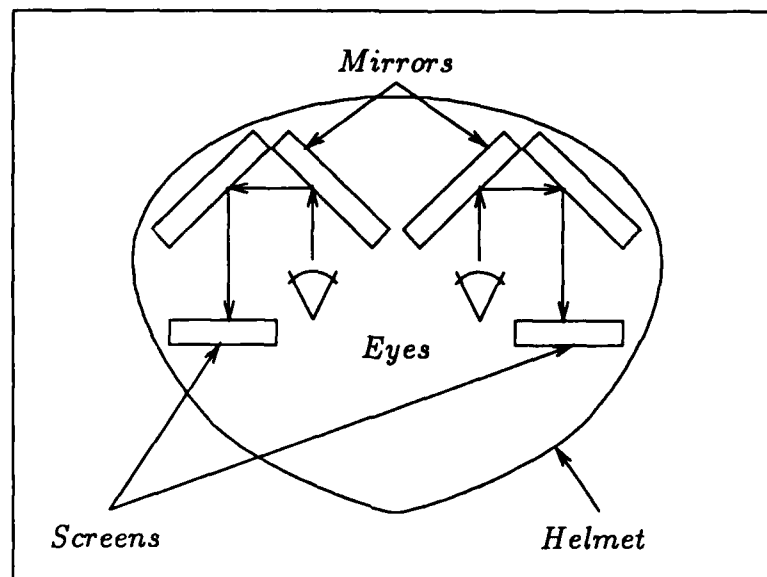


Figure 20. Top View of the Second HMD-II Design

3.3.3 Components. Several major components were used in the construction of the HMD-II. These include new optics, LCD color televisions, and the face seal from a scuba diving mask. Each of these components are described in the following sections.

3.3.3.1 Optics. The optics used in the AFIT HMD-II were designed by Eric Howlett and built by Pop-Optix Labs of Waltham, MA (see Figure 22). They were originally designed for use in a stereo slide viewer, but have been used in head-mounted displays before (7:78). The optics have a focal length of approximately 2.5 inches, and a horizontal and vertical field-of-view of 120 degrees.

Though they provide excellent image quality, the optics exhibit considerable pin-cushion distortion as shown in Figure 23. This distortion can be compensated for in software by pre-distorting the image, but this can be complicated when using

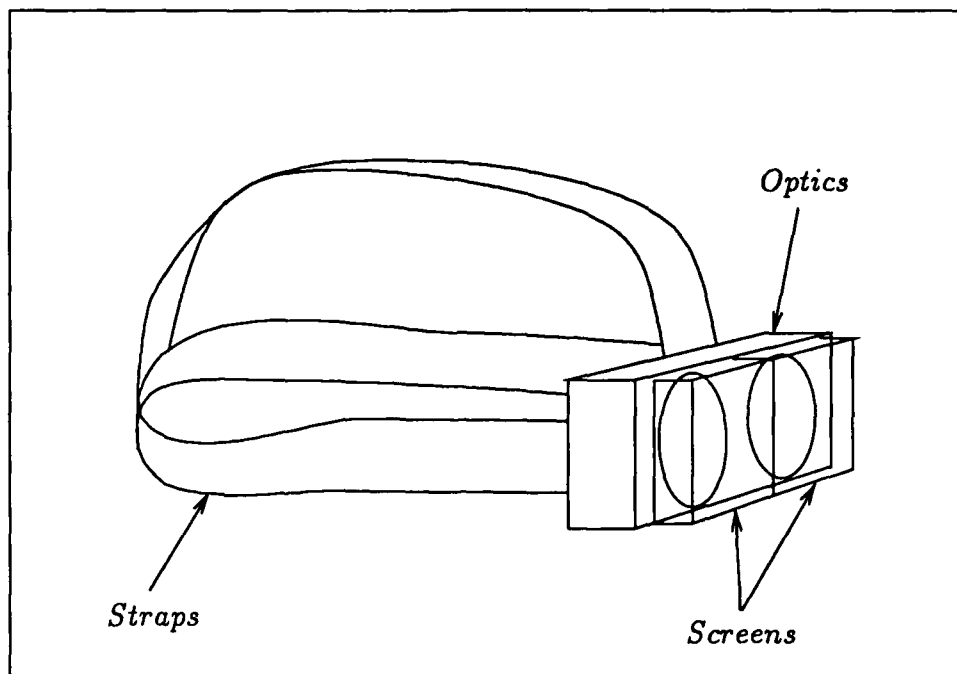


Figure 21. Final HMD-II Design

geometric primitives on a raster system, such as the software for this thesis does.

3.3.3.2 LCD Televisions. The display screens used in HMD-II are Sony FDL-330 color monitors. The FDL-330 is a three component monitor system with separate detachable power pack, tuner, and monitor sections (20). The monitors use active matrix thin film transistor technology to drive the liquid crystal display screens. The screens are 2.7 inches diagonal measure with over 86,000 pixels arranged in a 360×240 grid. A fluorescent backlight is used to brighten the image. The monitor section can accept direct video input so the tuner section has been removed to save weight. The Sony FDL-330 is shown in Figure 24.



Figure 22. LEEP Optics

3.3.3.3 Face Seal. For the system to work most effectively, the system must seal tightly against the wearer's face for a comfortable fit and total light blockage. The black silicon rubber skirt from a scuba diving mask works well for this application.

3.3.4 Construction. The construction of HMD-II involved much more than collecting a few components and connecting them together. Modifications to some components were required and all the pieces had to be integrated into a working, usable system.

3.3.4.1 Television Modification. The Sony FDL-330 monitors were almost perfect for this application. Only two minor modifications were necessary.

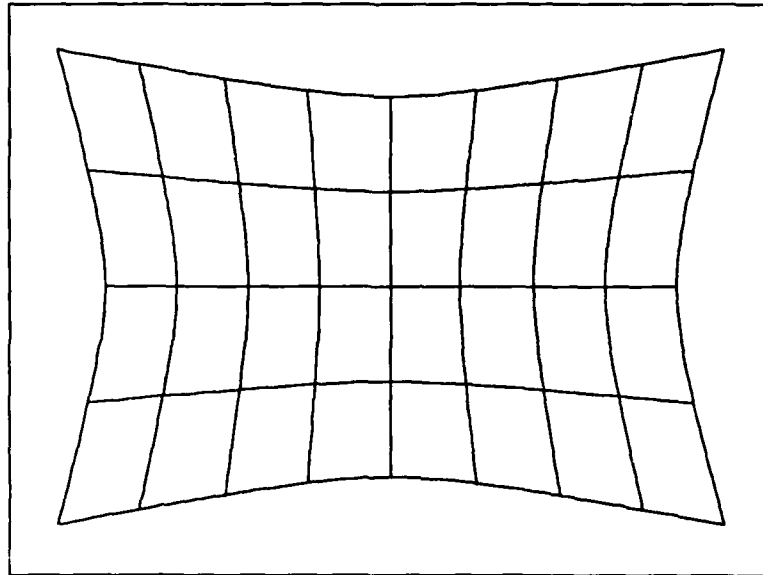


Figure 23. Pin-cushion Distortion of LEEP Optics

Due to the close proximity the screens were to each other, the audio/video input jack had to be moved from the side of the case to the bottom. Instead of moving the actual plug, the connector was bypassed by a wire through the bottom of the case. The +6.5V power input on the power pack presented the same problem as the audio/video input and was solved in the same manner.

3.3.4.2 Optics Modification. The LEEP optical elements are made of plastic and housed in a plastic case. They do, however, have a metal frame around them for mounting in the stereo viewer (see Figure 22 on Page 40). This metal frame, while adding unwanted weight, also interferes with the magnetic field of the Polhemus source, causing false readings. In HMD-II the frame has been removed and the optics incorporated directly into a fiber glass mount.

One other significant adjustment has been made to the LEEP optics. Since

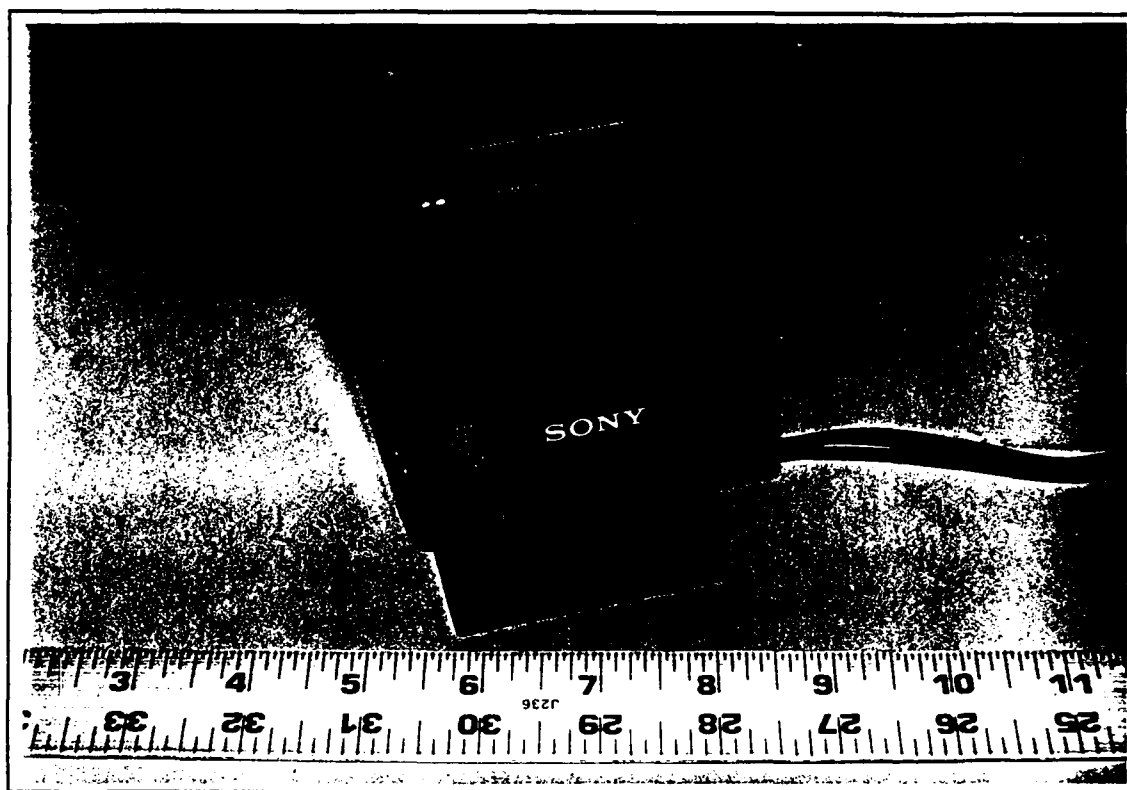


Figure 24. Sony FDL-330 Color LCD Monitor

the Sony monitors are slightly too big to align perfectly with the central axis of the LEEP optics, Fresnel press-on prisms are used to adjust the line of sight. Though the Fresnel prisms caused a great deal of distortion in the optical system of HMD-I, the prisms actually improve the image of HMD-II. The LEEP optics are so good, and the screen image magnified so much, that each individual screen pixel can be seen. The screen appears as a grid of disconnected pixels instead of coherent image. The prisms act as diffusers, smearing the image slightly, and making the image more coherent. In fact, VPL uses diffusers in their Eyephone head-mounted display for the same reason (10).

3.3.4.3 System Integration. Figure 25 shows the completed system. The scuba mask skirt has been attached to the fiber glass frame holding the optics. The Sony monitors are mounted on rails and can slide horizontally. The horizontal position of the screens can be changed by simply sliding the TVs outward. The horizontal movement of the screens permits adjustment for persons with different inter-pupillary distances. This is necessary so the two images will converge into one within the binocular overlap area. Vertical motion is not adjustable and has not found to be necessary. The whole system is enclosed in a removable fiber glass shell.

The system, when being worn, is held in place by three adjustable straps, one over the top of the head, and one to either side (see Figure 26). The straps are attached to a pad at the back of the head. The pad is cushioned and filled with lead shot to act as a counterweight. The strap across the top of the head has a Velcro mount for the Polhemus sensor, and mounts for the video and power cables to drive the system. The cables terminate in a 10-pin connector, which allows the system to be disconnected from its drive hardware.

3.4 Summary

The requirements for AFIT HMD-II, with respect to AFIT HMD-I, were to (1) improve image quality, (2) reduce system weight, and (3) make it easier to use. These requirements have been met. Using the LEEP optics, even with the press-on prisms, have greatly improved the image quality. The new Sony monitors have also helped because though they have the same pixel resolution as the Sharp TVs from HMD-I, they have a smaller screen resulting in an effective higher resolution. By removing the helmet from the system, total weight has been reduced. Even more weight could be removed if the television electronics could be separated from the display screens. The system is easy to use; slip it on, tighten the straps, and the system is ready to use.

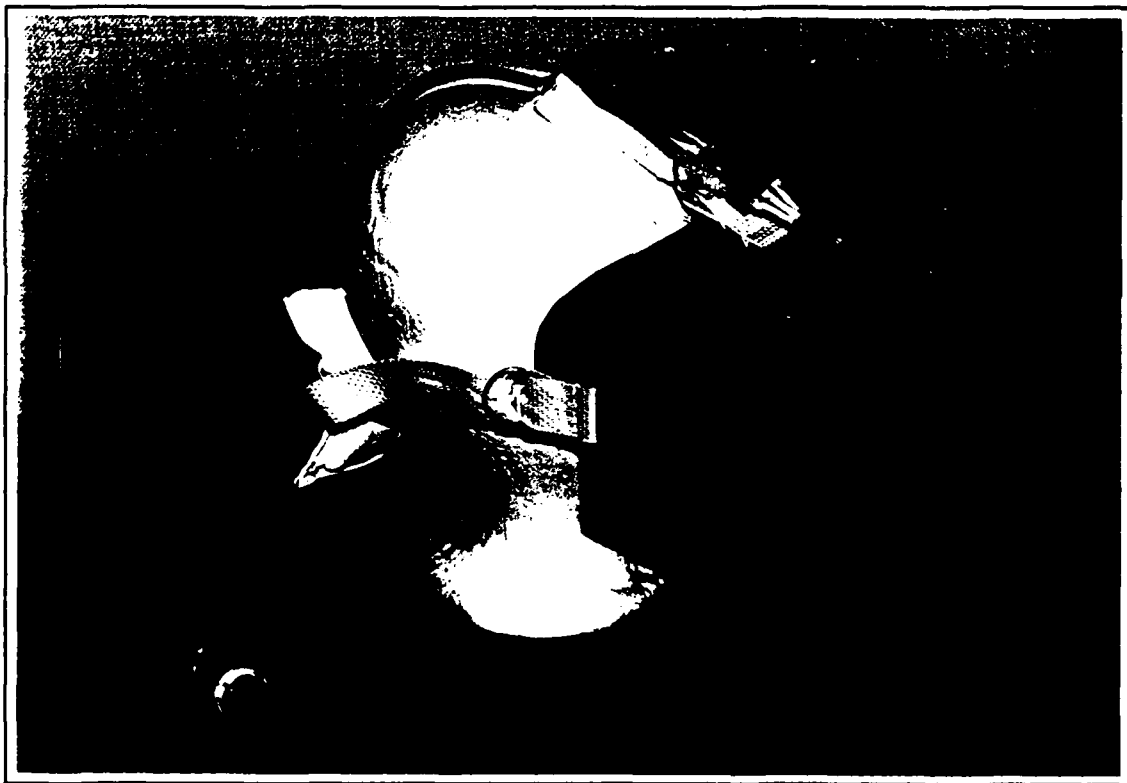


Figure 25. HMD-II

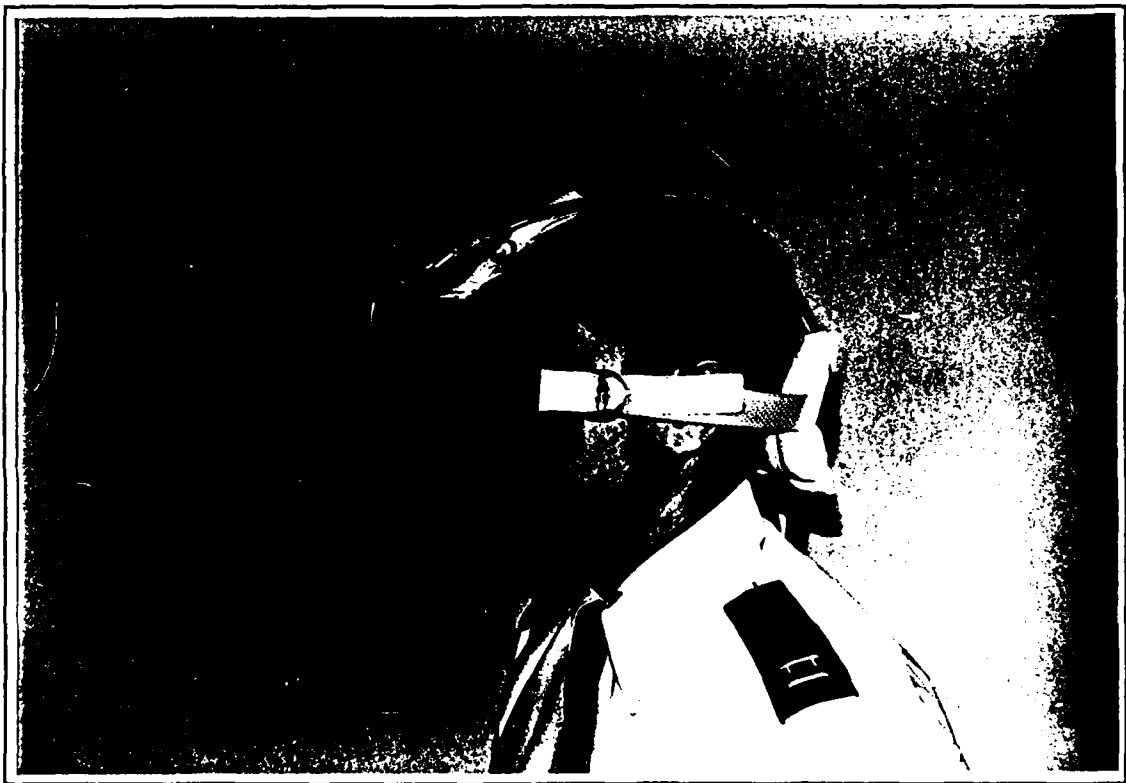


Figure 26. HMD-II In Use

IV. Summary

4.1 Results

As stated in the original scope of this thesis, the goal for this effort was to develop a 3-D virtual environment display system. The system would consist of a library of software routines specifically tailored for use in a virtual environment system, and a head-mounted display for placing the user in the virtual environment. That goal has been accomplished and the system works well, despite some minor problems.

4.1.1 Software. The following two sections describe the results of this thesis with regard to the VEDS software library.

4.1.1.1 Input Device Routines The original requirements for the input device routines were (a) to provide a standard interface across all devices, (b) to support four unique input devices, and (c) provide support for remote input. All of these requirements have been met.

The input routines for all devices have the same syntax and programming usage. Support is provided for a CH Products Microstick Joystick, a CIS Dimension Six Force-Torque Ball, a Polhemus 3-Space Tracker, and a VPL DataGlove. Each of these devices is also supported in a remote input mode; the devices need not be attached to the machine on which they are used.

4.1.1.2 Graphics Routines The original requirements for the graphics routines were (a) provide a library of routines to permit the construction and display of a virtual environment, (b) make the library easy to use and highly functional, and (c) maintain an update rate of ten frames per second. All these requirements have been met to a certain degree.

The graphics library contains enough functionality to build and display a virtual environment. The virtual environment is currently limited to wireframe and filled convex polygonal objects and a simple terrain model. These are sufficient to construct a simple virtual environment. The library has a high degree of functionality; as few as six library calls are necessary to build and display a virtual environment. The update rate is highly dependent upon the number of graphics primitives being displayed and can fluctuate greatly.

4.1.2 Head-Mounted Display. The original requirements for the HMD-II, with respect to the HMD-I, were (a) to improve the image quality, (b) to reduce the system weight, and (c) make the system easier to use and wear.

The results here are a mixed bag. The new LEEP optics have definitely improved the image quality over those in the HMD-I system, but the quality is not great. The optics magnify so well that each pixel on the screen is visible. The use of the press-on prisms help to blur, and improve, the image, but higher resolution screens are the correct solution. The system weight has been reduced some, but has been shifted completely to the front of the head; this requires the use of a counterweight to help balance the load. The system weight still needs to be reduced further.

4.2 Recommendations

No project is ever perfect, and this one is no exception. Though the original goal for the thesis has been met, much more remains to be done, and some things could be done differently.

4.2.1 Software. The following are recommended changes to the VEDS software library as it now exists.

1. Avoid the use of the Simgraphics code for handling DataGlove input. Though the DataGlove reading code works fine in its present configuration, some performance improvements could be gained by bypassing the Simgraphics library and talking directly to the DataGlove. This would also avoid having to run the Simgraphics DataGlove server, thus reducing the number of executing processes.
2. Improve the handling of button presses better. This might involve using a queue or software interrupt to insure that button presses are not missed. It is currently possible to miss a button press if it occurs between sampling events.
3. Provide a better terrain model. This might involve implementing a multi-level terrain database. Perhaps the same technique used in flight simulators for personal computers could be used.
4. Replace software with hardware. A computer capable of real-time Z-buffering could eliminate all the problems in the object display routines by completely solving the hidden surface problem.

4.2.2 Head-Mounted Display. The following are recommended changes to the HMD-II as it currently exists, and some suggestions for new display technology.

1. Further reduce the weight. The LCD screens must be removed completely from their electronics, and the electronics removed from the head-mounted display. The fiber glass shell holding the TVs is too bulky and heavy; a new lighter weight design will have to be devised.
2. Remove the prisms from the system. To do this will require either smaller TVs or custom made LCDs with two screens on one substrate. The smaller TVs are undesirable because they will not completely fill the field-of-view. However, the custom made LCDs will be prohibitively expensive.

3. Explore new display technologies. Display devices such as the Private Eye (16) and the Tektronix stereo display (26) should be investigated as alternatives to the head-mounted display.

4.3 Future Applications

Potential applications for virtual environment technology are almost limitless. Two specific applications being investigated at AFIT are (a) the replay of aircraft mission data, allowing the pilots and commanders to review the mission (12), and (b) the use of a virtual environment display for battle management and mission planning (28). As described in Section 1.5, many other applications of virtual environment displays have already been discovered. These range from flight simulators to molecular visualization to architectural walkthroughs to remote robotic control. And this appears to only be the tip of the iceberg for virtual environments. Medical imaging and air traffic control are already emerging as possible uses for a virtual environment system, and the potential for personal entertainment systems is unlimited.

Appendix A. *Manual Pages*

On the following pages are the Unix manual page descriptions of the software routines available in VEDS. Each page contains the C language declaration of the function, a brief explanation of what it does, and any special use guidelines.

NAME

VE_close_dataglove, VE_init_dataglove, VE_open_dataglove, VE_parse_dataglove, VE_read_raw_dataglove, VE_read_dataglove, VE_write_dataglove – VPL DataGlove Input Routines

SYNOPSIS

```
#include <VE.h>
void VE_close_dataglove()
void VE_init_dataglove(calibration_file, gesture_file)
String calibration_file;
String gesture_file;
void VE_open_dataglove(ttyport, speed)
String ttyport;
int speed;
void VE_parse_dataglove(dd, buf)
DatagloveData *dd;
char buf[];
int VE_read_raw_dataglove(buf, len)
char buf[];
int len;
void VE_read_dataglove(dd)
DatagloveData *dd;
void VE_write_dataglove(buf, len)
char buf[];
```

DESCRIPTION

These routines are used to interact with the VPL DataGlove.

VE_close_dataglove closes the tty port to the DataGlove. Closing a port not previously opened has undefined results.

VE_init_dataglove initializes the DataGlove. The *calibration_file* parameter is a String containing the null-terminated name of a DataGlove calibration file created with the Simgraphics DataGlove Test and Calibration software. If *calibration_file* is NULL, a fast calibration will be done. Fast calibration requires that the hand be held in three different positions while the DataGlove control unit samples the glove. The *gesture_file* parameter is a String containing the null-terminated name of a DataGlove gesture file created with the Simgraphics

DataGlove Gesture Editor. If *gesture_file* is NULL, no gestures can be recognized. Be sure the DIP switch settings on the back of the DataGlove control unit are set to 19200 baud, 2 stop bits, and RS-232 interface.

VE_open_dataglove opens the DataGlove for use. The DataGlove should be connected to the tty port specified by the null terminated string *ttyport*. The baud rate is specified by the *speed* parameter. The baud rate should be specified by a capital "B" followed by the baud rate as in "B9600". See the file */usr/include/termio.h* for a list of the baud rate codes. Be sure that the selected speed matches the DIP switch settings on the back of the DataGlove control unit. Note: the *speed* parameter is currently ignored since the Simgraphics DataGlove software used to read the glove only supports 19200 baud.

VE_parse_dataglove parses the raw data stream from the DataGlove into the *DatagloveData* structure pointed to by *dd*. The raw data record should be in *buf*, exactly as returned from *VE_read_raw_dataglove*.

VE_read_raw_dataglove reads data directly from the DataGlove placing it in the String pointed to by *buf*. The length of the data stream to read is specified by the parameter *len*. This routine will normally not be used.

VE_read_dataglove essentially runs *VE_read_raw_dataglove* and *VE_parse_dataglove* to fill the *DatagloveData* structure pointed to by *dd* with the most current data from the DataGlove. This is the normal way to get information from the DataGlove.

DATA TYPES

```
#define NUMSENSORS 10
typedef struct
{
    SmallPosition pos;
    Orientation or;
    short flex[NUMSENSORS];
    Byte gesture;
    char gesture_name[16];
} DatagloveData;
```

SEE ALSO

DataGlove Programmer's Toolkit Programmer's Guide, DataGlove Gesture Editor User's Manual, *remote(LOCAL)*, *server(LOCAL)*

DATAGLOVE(LOCAL) Virtual Environment Manual DATAGLOVE(LOCAL)

AUTHOR

Bob Filer

NAME

VE_close_joystick, VE_init_joystick, VE_open_joystick, VE_parse_joystick, VE_read_raw_joystick, VE_read_joystick, VE_write_joystick – CH Products Microstick Joystick Input Routines

SYNOPSIS

```
#include <VE.h>
void VE_close_joystick()
void VE_init_joystick(mode)
    Byte mode;
void VE_open_joystick(ttyport, speed)
    String ttyport;
    int speed;
void VE_parse_joystick(jd, buf)
    JoystickData *jd;
    char buf[];
int VE_read_raw_joystick(buf, len)
    char buf[];
    int len;
void VE_read_joystick(jd)
    JoystickData *jd;
void VE_write_joystick(buf, len)
    char buf[];
```

DESCRIPTION

These routines are used to interact with the CH Products Microstick Joystick.

VE_close_joystick closes the tty port to the Joystick. Closing a port not previously opened has undefined results.

VE_init_joystick initializes the Joystick to report data in a the format specified by the *mode* parameter. By default, the Joystick uses the JOY_RATE_ABS data format. Five additional data formats are available as shown in the following table.

JOY_RATE_ABS	Rate + Absolute Movement
JOY_ZOOM_ABS	Zoom + Absolute Movement
JOY_UNM_ABS	Unmapped Absolute Movement

JOY_ZOOM	Zoom Movement
JOY_RATE	Rate Movement
JOY_ABSOLUTE	Absolute Movement

Be sure the DIP switch settings on the Joystick match the *mode* selected. See the Microstick User's Guide for a full explanation of the six types of movement.

VE_open_joystick opens the Joystick for use. The Joystick should be connected to the tty port specified by the null terminated string *ttyport*. The baud rate is specified by the *speed* parameter. The baud rate should be specified by a capital "B" followed by the baud rate as in "B9600". See the file */usr/include/termio.h* for a list of the baud rate codes. Be sure that the selected speed matches the DIP switch settings on the bottom of the Joystick.

VE_parse_joystick parses the raw data stream from the Joystick into the *JoystickData* structure pointed to by *jd*. The raw data record should be in *buf*, exactly as returned from *VE_read_raw_joystick*.

VE_read_raw_joystick reads data directly from the Joystick placing it in the String pointed to by *buf*. The length of the data stream to read is specified by the parameter *len*. This routine will normally not be used.

VE_read_joystick essentially runs *VE_read_raw_joystick* and *VE_parse_joystick* to fill the *JoystickData* structure pointed to by *jd* with the most current data from the Joystick. This is the normal way to get information from the Joystick.

DATA TYPES

```
typedef struct
{
    short button;
    short x;
    short y;
} JoystickData;
```

SEE ALSO

CH Products Microstick User's Guide, *remote(LOCAL)*, *server(LOCAL)*

AUTHOR

Bob Filer

NAME

VE_close_polhemus, *VE_init_polhemus*, *VE_open_polhemus*, *VE_parse_polhemus*,
VE_read_raw_polhemus, *VE_read_polhemus*, *VE_write_polhemus* - Polhemus
3-Space Tracker Input Routines

SYNOPSIS

```
#include <VE.h>
void VE_close_polhemus()
void VE_init_polhemus()
void VE_open_polhemus(ttyport, speed)
String ttyport;
int speed;
void VE_parse_polhemus(pd, buf)
PolhemusData *pd;
char buf[];
int VE_read_raw_polhemus(buf, len)
char buf[];
int len;
void VE_read_polhemus(pd)
PolhemusData *pd;
void VE_write_polhemus(buf, len)
char buf[];
void VE_correct_polhemus_for_lab(pd)
PolhemusData *pd;
```

DESCRIPTION

These routines are used to interact with the Polhemus 3-Space Tracker.

VE_close_polhemus closes the tty port to the Polhemus. Closing a port not previously opened has undefined results.

VE_init_polhemus initializes the Polhemus to report data in a format specific to the VEDS and the Polhemus' location in the graphics lab. To do this, the Polhemus is first cold reset (a loud beep can be heard when this happens). Next the Polhemus boresight is set to default, increment to 0, ASCII format, and units in inches. The Polhemus operational envelope is set to the maximum 65 inches in all directions. Next the operational hemisphere is set to have its

zenith on the Z axis (pointing towards the floor in the graphics lab). The output record is then set to the following:

Code	Value
2	Cartesian Coordinates (X, Y, Z)
4	Orientation Angles (azimuth, elevation, roll)
5	X-axis direction cosines (x, y, z)
6	Y-axis direction cosines (x, y, z)
7	Z-axis direction cosines (x, y, z)
1	CR, LF

The Polhemus is then set for continuous output mode. See the Polhemus 3-Space User's Manual for a full explanation of the options set by *VE_init_polhemus*.

VE_open_polhemus opens the Polhemus for use. The Polhemus should be connected to the tty port specified by the null terminated string *ttyport*. The baud rate is specified by the *speed* parameter. The baud rate should be specified by a capital "B" followed by the baud rate as in "B9600". See the file */usr/include/termio.h* for a list of the baud rate codes. Be sure that the selected speed matches the DIP switch settings on the back of the Polhemus.

VE_parse_polhemus parses the raw data stream from the Polhemus into the *PolhemusData* structure pointed to by *pd*. The raw data record should be in *buf*, exactly as returned from *VE_read_raw_polhemus*.

VE_read_raw_polhemus reads data directly from the Polhemus placing it in the String pointed to by *buf*. The length of the data stream to read is specified by the parameter *len*. This routine will normally not be used.

VE_read_polhemus essentially runs *VE_read_raw_polhemus* and *VE_parse_polhemus* to fill the *PolhemusData* structure pointed to by *pd* with the most current data from the Polhemus. This is the normal way to get information from the Polhemus.

VE_correct_polhemus_for_lab makes corrections to the polhemus data based on the location of the polhemus source within the graphics lab. This alters the *y* and *z* position by setting *y* = -*y* and *z* = 79 - *z*. The *y* and *z* direction cosines are also inverted.

DATA TYPES

```
typedef struct
{
    Byte sensor;
```

```
    Position pos;  
    Orientation or;  
    DirectionCosine xdc;  
    DirectionCosine ydc;  
    DirectionCosine zdc;  
    Quaternion quat;  
} PolhemusData;
```

SEE ALSO

Polhemus 3-Space User's Manual, **remote(LOCAL)**, **server(LOCAL)**

AUTHOR

Bob Filer

NAME

VE_close_spaceball, *VE_init_spaceball*, *VE_open_spaceball*, *VE_parse_spaceball*,
VE_read_raw_spaceball, *VE_read_spaceball*, *VE_write_spaceball* – CIS Dimension Six Spaceball Input Routines

SYNOPSIS

```
#include <VE.h>
void VE_close_spaceball()
void VE_init_spaceball(mode)
Byte mode;
void VE_open_spaceball(ttyport, speed)
String ttyport;
int speed;
void VE_parse_spaceball(sd, buf)
SpaceballData *sd;
char buf[];
int VE_read_raw_spaceball(buf, len)
char buf[];
int len;
void VE_read_spaceball(sd)
SpaceballData *sd;
void VE_write_spaceball(buf, len)
char buf[];
```

DESCRIPTION

These routines are used to interact with the CIS Dimension Six Spaceball.

VE_close_spaceball closes the tty port to the Spaceball. Closing a port not previously opened has undefined results.

VE_init_spaceball initializes the Spaceball to report data in a the format specified by the *mode* parameter. By default, the Spaceball uses the SB_FORCE data format. The other available format is SB_VOLTAGE. Be sure the DIP switch settings on the Spaceball match the *mode* selected. See the Dimension Six User's Guide for a full explanation of the two data formats.

VE_open_spaceball opens the Spaceball for use. The Spaceball should be connected to the tty port specified by the null terminated string *ttyport*. The

baud rate is specified by the *speed* parameter. The baud rate should be specified by a capital "B" followed by the baud rate as in "B9600". See the file `/usr/include/termio.h` for a list of the baud rate codes. Be sure that the selected speed matches the DIP switch settings on the bottom of the Spaceball.

VE_parse_spaceball parses the raw data stream from the Spaceball into the *SpaceballData* structure pointed to by *sd*. The raw data record should be in *buf*, exactly as returned from *VE_read_raw_spaceball*.

VE_read_raw_spaceball reads data directly from the Spaceball placing it in the String pointed to by *buf*. The length of the data stream to read is specified by the parameter *len*. This routine will normally not be used.

VE_read_spaceball essentially runs *VE_read_raw_spaceball* and *VE_parse_spaceball* to fill the *SpaceballData* structure pointed to by *sd* with the most current data from the Spaceball. This is the normal way to get information from the Spaceball.

DATA TYPES

```
typedef struct
{
    short xtrans;
    short ytrans;
    short ztrans;
    short xrot;
    short yrot;
    short zrot;
    short button;
} SpaceballData;
```

SEE ALSO

Dimension Six User's Guide, `remote(LOCAL)`, `server(LOCAL)`

AUTHOR

Bob Filer

NAME

VEserver – Virtual Environment Input Server

SYNOPSIS

VEserver [*-f config_file*] [*-d*] &

DESCRIPTION

VEserver is the Virtual Environment Input Server. It has two main tasks: (1) continuously sample the configured input devices and (2) service network requests for input data. The server should be run in the background.

The operation of the server is controlled through a configuration file. This file specifies which input devices are connected, to which port, and at what baud rate. The configuration file also specifies the calibration and gesture files used with the VPL DataGlove.

By default, the configuration file *VEdefault.config* is used for configuration information. This may be overridden using the *-f* option. The format of the configuration file is shown in the following table.

Keyword	Path	Baud Rate
joystick	/dev/ttyXX	300-19200
spaceball	/dev/ttyXX	300-19200
polhemus	/dev/ttyXX	300-19200
dataglove	/dev/ttyXX	300-19200
calibration_file	/path/file	
gesture_file	/path/file	

SEE ALSO

remote(LOCAL)

AUTHOR

Bob Filer

NAME

VE_open_remote, VE_read_remote, VE_close_remote – Virtual Environment Remote Input Routines

SYNOPSIS

```
#include <VE.h>
void VE_open_remote(host)
String host;
void VE_read_remote(command, data)
Byte command;
Byte *data;
void VE_read_raw_remote(command, buf, size)
Byte command;
Byte *buf;
int size;
void VE_close_remote()
```

DESCRIPTION

These routines are used to get data from input devices attached to a remote machine. The remote machine must be running the *Virtual Environment Input Server* as explained in server(LOCAL) and have input devices attached. The currently supported input devices are the VPL Dataglove, the CH Products Microstick Joystick, the CIS Dimension Six Spaceball, and the Polhemus 3-Space Tracker.

VE_open_remote opens a connection to the server located on the machine specified by the null terminated string *host*.

VE_read_remote retrieves input data from the server for the device specified in *command*. The current valid values of *command* and their meanings are shown in the following table.

SEND_DATAGLOVE	DatagloveData
SEND_JOYSTICK	JoystickData
SEND_POLHEMUS1	PolhemusData for sensor 1
SEND_POLHEMUS2	PolhemusData for sensor 2
SEND_SPACEBALL	SpaceballData

The argument *data* should point to a structure of the appropriate type for the data being requested (see the table above).

VE_read_raw_remote reads the raw byte stream directly from the server placing it in *buf*. The length of the byte stream to read is specified by the parameter *size*. The *command* parameter is as shown in the preceding table. This routine will normally not be used.

VE_close_remote sends a SHUTDOWN command to the server and severs the connection.

EXAMPLE

The following sample code establishes a connection to the server, reads the current value from the Spaceball into *sd*, and then closes the connection.

```
#include <VE.h>
SpaceballData sd;
main()
{
    VE_open_remote("louvre");
    VE_read_remote(SEND_SPACEBALL, &sd);
    VE_close_remote();
}
```

SEE ALSO

dataglove(LOCAL), *joystick(LOCAL)*, *polhemus(LOCAL)*, *spaceball(LOCAL)*, *server(LOCAL)*

AUTHOR

Bob Filer

NAME

VE_lookat, *VE_perspective*, *VE_find_color* – Virtual Environment Graphics Control Routines

SYNOPSIS

```
#include <VE.h>
void VE_lookat(vx, vy, vz, px, py, pz, twist)
float vx, vy, vz, px, py, pz, twist;
void VE_perspective(fov, aspect, near, far)
float fov, aspect, near, far;
Colorindex VE_find_color(r, g, b)
float r, g, b;
```

DESCRIPTION

VE_lookat defines the viewpoint and a reference point on the line of sight in world coordinates with the Z axis up. The viewpoint is at (*vx*, *vy*, *vz*). The viewpoint and reference point (*px*, *py*, *pz*) define the line of sight. *Twist* measures the right-hand rotation about the Z-axis in the eye coordinate system.

VE_perspective defines a projection transformation by indicating the field-of-view angle *fovy* in the *y* direction of the eye coordinate system; the *aspect* ratio that determines the field of view in the *x* direction; and the distance to the *near* and *far* clipping planes in the *z* direction. The aspect ratio is a ratio of *x* to *y*. In general, the aspect ratio in *VE_perspective* should match the aspect ratio of the associated viewport. For example, *aspect* = 2.0 means the viewer's angle of view is twice as wide in *x* as it is in *y*. If the viewport is twice as wide as it is tall, it displays the image without distortion. *near* and *far* are the distances from the viewer to the near and far clipping planes, and are always positive.

VE_find_color returns a *Colorindex* into the color table corresponding to the RGB value specified by *r*, *g*, and *b*. The RGB values should be in the range $0.0 < \text{RGB} < 1.0$.

AUTHOR

Bob Filer

NAME

VE_display_hand - Virtual Environment Hand Display Routine

SYNOPSIS

```
#include <VE.h>
void VE_display_hand(data)
DatagloveData *data;
```

DESCRIPTION

The *VE_display_hand* routine is used to display a wireframe model of a human right hand. The *data* parameter is a pointer to a *DatagloveData* structure which should contain values indicating position, orientation, and finger flex of the hand.

The hand is modeled with the palm centered at the origin. The palm is 4h(Z) x 3.75w(X) x 0.75d(Y). The fingers and thumb are 4h(Z) x 0.75w(X) x 0.75d(Y).

SEE ALSO

dataglove(LOCAL)

AUTHOR

Bob Filer

NAME

VE_create_menu, *VE_menu_choice*, *VE_menu_on*, *VE_menu_off* – Virtual Environment Menu Routines

SYNOPSIS

```
#include <VE.h>
VE_Object *VE_create_menu(text)
char *text[];
void VE_menu_choice(menu, choice)
VE_Object *menu;
Byte choice;
void VE_menu_on(menu)
VE_Object *menu;
void VE_menu_off(menu)
VE_Object *menu;
```

DESCRIPTION

The *VE_create_menu* builds a 3-D menu. The menu items are taken from *text*, which should be an array of pointers to null-terminated strings. Each string will be one menu choice. A NULL pointer ends the list.

VE_menu_choice highlights the menu item specified by *choice*. Menu items begin numbering at 1. *Choice* should be set to NO_CHOICE when no highlighting is desired.

VE_menu_on and *VE_menu_off* turn the display of the menu on and off.

EXAMPLE

The following code fragment will build a menu containing the choices "Start" and "Quit":

```
char *text[] = {"Start", "Quit", NULL};
VE_Object *menu;
menu = VE_create_menu(text);
```

AUTHOR

Bob Filer

NAME

VE_build_object, VE_display_object, VE_display_objects, VE_translate_object,
VE_rotate_object, VE_scale_object – Virtual Environment Object Routines

SYNOPSIS

```
#include <VE.h>
VE_Object *VE_build_object(filename)
String filename;
void VE_display_object(veobj, eye, lookat)
VE_Object *veobj;
Position *eye;
Position *lookat;
void VE_display_objects(objects, size, eye, lookat)
VE_Object *objects[];
int size;
Position *eye;
Position *lookat;
void VE_translate_object(obj, x, y, z, action)
VE_Object *obj;
float x, y, z;
Byte action;
void VE_rotate_object(obj, x, y, z, action)
VE_Object *obj;
float x, y, z;
Byte action;
void VE_scale_object(obj, x, y, z, action)
VE_Object *obj;
float x, y, z;
Byte action;
```

DESCRIPTION

VE_build_object takes a AFIT geometry file description of an object and builds a Virtual Environment Object (*VE_Object*). The parameter *filename* should be a null-terminated string containing the name of an AFIT geometry file. Both a wireframe and a BSP-tree representation are built.

VE_display_object displays the object pointed to by *veobj* based on the eye point specified by *eye* and the lookat point specified by *lookat*.

VE_display_objects displays all the objects contained in the array of pointers to objects *objs*. The number of objects in the array is specified by the *size* parameter. The view is based on the eye point *eye* and the lookat point *lookat*.

VE_translate_object, *VE_rotate_object*, and *VE_scale_object* translate, rotate, and scale the Virtual Environment Object pointed to by *obj*. The amount of the transformation in each direction is specified by the values *x*, *y*, and *z*. The *action* parameter indicates how the transformation values will be applied to the current transformation values for the object. Possible values for *action* and the associated action are shown below.

XFORM_SUBTRACT	New values are subtracted from the current values
XFORM_ADD	New values are added to the current values
XFORM_EQUAL	Current values are replaced by the new values
XFORM_REPLACE	Current values are replaced by non-zero new values

DATA TYPES

```
typedef struct
{
    Colorindex color;
    Boolean visible;
    SmallPosition scale;
    SmallPosition rotation;
    Position position;
    float distance;
    Position centroid;
    Position bb[BBOXSIZE];
    Displaytype type;
    Boolean display;
    Object wire;
    TreeNode *bsp;
    Menu menu;
    Matrix matrix;
    Matrix inverse_matrix;
} VE_Object;
```

AUTHOR

Bob Filer

NAME

VE.build_terrain - Virtual Environment Terrain Routines

SYNOPSIS

```
#include <VE.h>
Object VE_build_terrain(minx, miny, maxx, maxy, z)
float minx, miny, maxx, maxy, z;
```

DESCRIPTION

VE_build_terrain creates a single green polygon (a rectangle) with corners specified by the *minx*, *miny*, *maxx*, and *maxy* parameters. The terrain is parallel to the X-Y plane at a height specified by *z*. The green terrain is overlaid with a black 10 x 10 grid.

The terrain should be displayed using the Silicon Graphics routine **callobj**, and should be called before displaying any other objects.

AUTHOR

Bob Filer

NAME

VE_begin, *VE_begin_graphics*, *VE_end*, *VE_end_graphics*, *VE_abort* – Virtual Environment Control Routines

SYNOPSIS

```
#include <VE.h>
void VE_begin()
void VE_begin_graphics(screen_mode)
int screen_mode;
void VE_end()
void VE_end_graphics()
void VE_abort(msg)
String msg;
```

DESCRIPTION

VE_begin initializes the Virtual Environment Display System (VEDS). This should be the first routine called in any VEDS program.

VE_begin_graphics initializes the graphics subsystem of the VEDS. This should be the second routine called in any VEDS program which uses graphics. The *mode* parameter should be either NTSC or HZ60 depending on whether NTSC or 60 hertz mode is desired.

VE_end shuts down the VEDS. This should be the last routine called in any VEDS program.

VE_end_graphics shuts down the graphics subsystem of the VEDS. This should be the next to last routine called in any VEDS program which uses graphics.

VE_abort causes an immediate but graceful abort from the VEDS. The parameter *msg* should be a null-terminated string which will be printed to standard output upon exit from the VEDS.

AUTHOR

Bob Filer

NAME

VE_close_geometryfile, VE_open_geometryfile, VE_read_geometryfile, VE_get_face
– Virtual Environment Geometry File Reading Routines

SYNOPSIS

```
#include <VE.h>
void VE_close_geometryfile()
void VE_open_geometryfile(filename)
String filename;
void VE_read_geometryfile()
Face *VE_get_face()
```

DESCRIPTION

These routines are used for reading AFIT geometry files with the Virtual Environment Display System.

VE_close_geometryfile closes a previously opened geometry file. Calling *VE_close_geometryfile* without previously opening a geometry file is an error.

VE_open_geometryfile opens the file specified by the null-terminated string *filename* which should be the name of an AFIT geometry file.

VE_read_geometryfile reads the previously opened geometry file into an internal data structure making it available for access through calls to *VE_get_face*. Subsequent calls to *VE_get_face* return pointers to *Face* structures for each polygon described in the AFIT geometry file. *VE_get_face* returns NULL when all polygons have been exhausted.

SEE ALSO

AFIT Geometry File Format Specification

AUTHOR

Bob Filer

NAME

VE_get_identity_matrix, VE_copy_matrix, VE_mult_matrix, VE_invert_matrix,
VE_mult_vec_matrix, VE_mult_vecstruct_matrix, VE_dot_vector, VE_dot_vectorstruct,
VE_cross_vector, VE_normalize_vector, VE_copy_vector, VE_add_vector, VE_subtract_vector,
VE_equal_vector, VE_distance – Virtual Environment Matrix and Vector Math
Routines

SYNOPSIS

```
#include <VE.h>
void VE_get_identity_matrix(m)
Matrix m;
void VE_copy_matrix(m1, m2)
Matrix m1, m2;
void VE_multiply_matrix(result, m1, m2)
Matrix result, m1, m2;
void VE_invert_matrix(orig, inv)
Matrix orig, inv;
void VE_mult_vec_matrix(result, v, m)
Vector result, v; Matrix m;
void VE_mult_vecstruct_matrix(result, v, m)
VectorStruct *result, *v; Matrix m;
double VE_dot_vector(v1, v2)
Vector v1, v2;
double VE_dot_vectorstruct(v1, v2)
VectorStruct *v1, *v2;
void VE_cross_vector(result, v1, v2)
Vector result, v1, v2;
double VE_normalize_vector(v)
Vector v;
void VE_copy_vector(v1, v2)
Vector v1, v2;
void VE_add_vector(result, v1, v2)
Vector result, v1, v2;
void VE_subtract_vector(result, v1, v2)
Vector result, v1, v2;
Boolean VE_equal_vector(v1, v2)
Vector v1, v2;
```

double VE_distance(from, to)
Position *from, *to;

DESCRIPTION

These routines are used for simple vector and matrix manipulations.

VE_get_identity_matrix sets *m* to the identity matrix.

VE_copy_matrix copies matrix *m1* into *m2*.

VE_mult_matrix computes the matrix multiplication $\text{result} = [m1][m2]$.

VE_invert_matrix computes the inverse of *orig* and places it in *inv*.

VE_mult_vec_matrix and *VE_mult_vecstruct_matrix* compute the vector-matrix multiplication $\text{result} = [v][m]$.

VE_dot_vector and *VE_dot_vectorsstruct* compute the vector dot product of *v1* and *v2*.

VE_cross_vector computes the vector cross product of *v1* into *v2*. *VE_normalize_vector* normalizes the vector *v* and returns the length of the original vector.

VE_copy_vector copies *v1* into *v2*.

VE_add_vector adds the vectors *v1* and *v2*.

VE_subtract_vector subtracts the vector *v2* from *v1*. *VE_equal_vector* compares the vectors *v1* and *v2* element by element returning TRUE if they are the same.

VE_distance computes the Euclidean distance between the 3-space points *from* and *to*.

AUTHOR

Bob Filer

Appendix B. *HMD-II User's Guide*

Introduction

The AFIT HMD-II is a fully enclosed head-mounted display system using wide-angle optics and color LCD televisions. The LCD televisions are powered by an external power supply. The system has two separate video input channels and is capable of displaying a stereo image. The video signals and external power are brought to the HMD-II system through an external cable bundle, which can be detached from the drive electronics.

The following sections explain how to setup and operate the AFIT HMD-II. In addition, the construction of the cable bundle is explained in detail.

Preparing the HMD-II for Use

Before the HMD-II can be used it must be attached to its drive electronics. The drive electronics consist of an adjustable voltage power supply and a Lyon-Lamb RGB—NTSC converter. The HMD-II is connected to the drive electronics with a 20 foot *tether* cable. Figure 27 shows a block diagram of the complete HMD-II system.

The following procedure should be used when attaching the HMD-II to the drive electronics.

1. Begin by only attaching the 20 foot *tether* cable to the drive electronics; the HMD-II should be detached from the *tether* at the 10-pin connector. Ensure that both the power supply and the Lyon-Lamb encoder are turned off before proceeding.
2. Attach the brown power cord to the power supply. The cord is color-coded with red indicating the positive (+) lead.

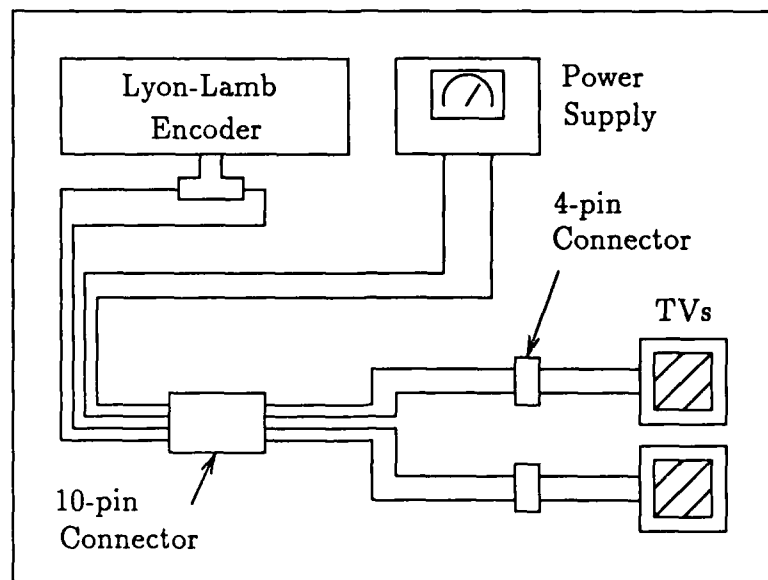


Figure 27. Block Diagram of the HMD-II System

3. Attach the left and right video cables to the Lyon-Lamb encoder using a "T" connector. The "T" connector should be attached to the *Video Out 1* connector on the back of the Lyon-Lamb.
4. Turn on the power supply and adjust the voltage to read +6 volts. **The HMD-II should not be attached while adjusting the power supply.** Once adjusted, turn off the power supply.
5. Attach the HMD-II *pigtail* cable to the *tether* cable using the 10-pin connector. Screw the two halves of the connector together to secure the connection.
6. Turn on the power supply and the Lyon-Lamb encoder. The system is now ready for use.

HMD-II Enclosure Maintenance

The AFIT HMD-II is essentially maintenance-free. There are almost no moving parts, and very few adjustable parts. However, two particular elements of the AFIT HMD-II require further explanation. These are the cable bundle attaching the HMD-II to its drive electronics, and the press-on prisms.

Cable Bundle The HMD-II cable bundle consists of two sections joined together at a 10-pin connector. The main section of the cable, the *tether*, is approximately 20 feet long. It carries power and video from the power supply and the Lyon-Lamb RGB—NTSC encoder to the 10-pin connector. The second segment of cable, the *pigtail*, is only about 3 feet long. It carries power and video from the 10-pin connector to the televisions inside the HMD-II. The *pigtail* is permanently attached to the head-mounted display.

Cable Specifications. In the *tether* segment, power is carried over 16 gauge, two conductor extension cord wire. The end connected to the power supply is color-coded; red indicates the positive (+) lead. The video signal is carried over RG59 coaxial cable. There is a separate cable for the left and right video signals for stereo support.

In the *pigtail* segment, power is carried over RG174 coaxial cable. The center conductor is the positive (+) lead. The video signal is carried over RG58 coaxial cable. Once again there is a separate cable for the left and right video signals.

Pinouts. Table 6 and Figure 28 show the pinouts for the *tether* and *pigtail* segments of the HMD-II wiring harness. For example, referring to Table 6, the positive power lead comes from the power supply to pin 9 in the 10-pin connector. This lead is then split into two separate wires, each running to pin 4 of the left and right 4-pin connectors.

Table 6. HMD-II Wiring Harness Pinouts

Drive Equipment	10-pin Connector	4-pin Connectors	Televisions
1 Positive (+) Lead	9	4L and 4R	Power
2 Negative (-) Lead	8	3L and 3R	Ground
3 Right Video Conductor	2	2R	V. IN
4 Right Video Shield	1	1R	GND
5 Left Video Conductor	6	2L	V. IN
6 Left Video Shield	7	1L	GND

Fresnel Press-on Prisms. The Fresnel press-on prisms used in the AFIT HMD-II were purchased from Bell Optical of Dayton, Ohio. Each lens in the LEEP optics is fitted with a +25 diopter prism which has been cut to size. Thirty diopter prisms are available and should be used if someone with a narrow-set eyes is to use the system.

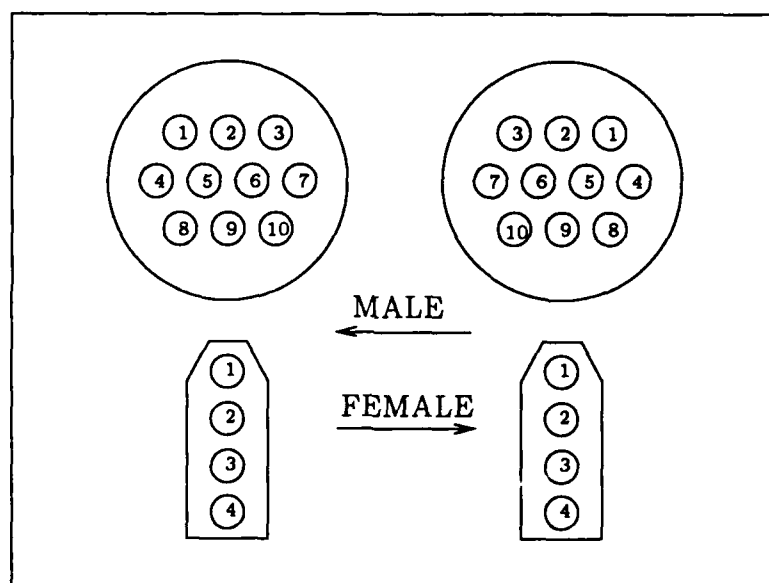


Figure 28. 10-Pin and 4-Pin Connector Pin Numbers

Appendix C. *AFIT Geometry File Format*

The AFIT geometry file provides a method for describing three dimensional objects composed of planar polygons, and for specifying attributes such as color and a shading model. The file is organized in a position dependent manner such that the position of a line within the file (its "line number") determines the class of information a line may contain.

The file syntax is shown in Table 7. In the table, literal symbols appear as contiguous strings of alphabetic characters. Substitutable symbols appear in angle brackets "<" and ">". Optional information appears within square brackets "[" and "]". C-style comments (*/* */*) and *#include* and *#define* directives may be contained within an AFIT geometry file.

Table 7. AFIT Geometry File Format

Line	Keywords	Comments
1	None	Up to 1024 characters
2	[ccw] [cw] [purge] [nopurge]	Geometry Parameters
3+	points <# of points> patches <# of patches> [parts <# of parts> <list> [bsp <# of nodes> [attributes <# of attributes> [textures <# of textures>]	Object component and attribute counts
4+	<x> <y> <z> [normal <i> <j> <k> [color <r> <g> [tindex <u> <v>]	Vertex Lines
5+	<n> <pt 1>...<pt n> [attribute <n> [texture <n> [type {PLAIN, COLOR, TEXTURE}]	Polygon/Patch Lines
6+	[shading {FLAT, GOURAUD, PHONG}] [reflectance {FLAT, PHONG, COOK}] [kd <n> [ks <n> [n <n> [opacity <n> [color <r> <g> [in <n> [material <filename>]	Attribute Lines
7+	filename	Texture Map Filename


Bibliography

1. Breglia, Dennis R. and others. "Helmet-mounted Laser Projector." In *Proceedings of the Image Generation/Display Conference II*, pages 241-258, Williams AFB AZ: Operations Training Division, Air Force Human Resources Laboratory, 1981.
2. Brooks, Frederick P. Jr. *Grasping Reality Through Illusion*. Technical Report TR88-007, University of North Carolina at Chapel Hill, March 1988.
3. Callahan, M. A. *A 3-D Display Head-Set for Personalized Computing*. MS thesis, Department of Architecture, Massachusetts Institute of Technology, 1983.
4. CH Products. *CH Products Microstick User's Guide*. Technical Report. San Marcos CA, 1985.
5. Chung, James C. and others. "Exploring Virtual Worlds With Head-Mounted Displays." In *Three Dimensional Visualization of Scientific Data, SPIE Proceedings*, Volume 1083, July 1989.
6. CIS Graphik und Bildverarbeitung GmbH. *Dimension 6 User's Manual*. Technical Report. Viersen West Germany, December 1987.
7. Fisher, Scott S. "Virtual Environment Display System." In *Proceedings of the 1986 Workshop on Interactive 3-D Graphics*, pages 77-87, New York: Association for Computing Machinery, 1987.
8. Fuchs, Henry and others. "On Visible Surface Generation by A Priori Tree Structures." In *Proceedings of the SIGGRAPH 80 Conference*, pages 124-133, New York: Association for Computing Machinery, July 1980.
9. Fuchs, Henry and others. "Near Real-Time Shaded Display of Rigid Objects." In *Proceedings of the SIGGRAPH 83 Conference*, pages 65-69, New York: Association for Computing Machinery, July 1983.
10. Grimaud, Jean-Jacques. Personal interview. Boston MA, 3 August 1989.
11. Leffler, Smauel J. and others. *An Advanced 4.3BSD Interprocess Communication Tutorial*. Technical Report, Berkeley CA: Computer Systems Research Group, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1986.
12. Lorimor, Gary Kim. *Real-Time Display of Time Dependent Data Using a Head-Mounted Display*. MS thesis, AFIT/GE/ENG/88D-22, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1988.
13. Polhemus Navigational Sciences Division, McDonnell Douglas Electronics Company. *3Space User's Manual*. Technical Report OPM3016-004. Colchester VT, January 1985.

14. Ponting, Bob. "Virtual Reality System Readied," *Infoworld*, 11:17+ (July 1989).
15. Rebo, Capt Robert K. *A Helmet-Mounted Virtual Environment Display System*. MS thesis, AFIT/GCS/ENG/88D-17, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1988.
16. Reflection Technology. *Introducing the Private Eye*. Product Announcement. Cambridge MA, August 1988.
17. Schachter, Bruce J. *Computer Image Generation*. New York: John Wiley and Sons, 1983.
18. Silicon Graphics Inc. *Silicon Graphics Pipeline*. Quarterly 007-7086-010. Mountain View CA, 1986.
19. Smith, Douglas B. and Dale G. Streyle. *An Inexpensive Real-Time Interactive Three-Dimensional Flight Simulation System*. MS thesis, Naval Postgraduate School, Monterey CA, June 1987.
20. Sony Corporation of America. *FDL-330 Color Watchman*. Product Announcement. Park Ridge NJ, March 1989.
21. Sun Microsystems Inc. *Sun-4/260*. Product Announcement. Mountain View CA, January 1988.
22. Sutherland, Ivan E. "The Ultimate Display." In *Proceedings of the IFIP Congress 65*, pages 506-508, 1965.
23. Sutherland, Ivan E. "A Head-Mounted Three Dimensional Display." In *Proceedings of the AFIPS Fall Joint Computer Conference*, pages 757-764, 1968.
24. Sutherland, Ivan E. and others. "A Characterization of Ten Hidden-Surface Algorithms," *Computing Surveys*, 6(1) (1974).
25. Task, H. Lee. Personal interview. Air Force Institute of Technology (AU), Wright-Patterson AFB OH, November 1988.
26. Tektronix, Inc. *Multi-Mode Stereoscopic 3-Dimensional Color Display*. Product Announcement. Beaverton OR, July 1989.
27. VPL Research, Inc. *Dataglove Model 2*. Product Announcement. Redwood City CA, March 1989.
28. Wardin, Capt Charles R. *Battle Management Visualization System*. MS thesis, AFIT/GE/ENG/89D-56, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1989.

Vita

Captain Robert E. Filer



Following high school, he attended the United States Air Force Academy where he received his Bachelor of Science degree in May 1984. After receiving his commission, he was assigned to the System Software Branch of the Information Systems Directorate at the Air Force Institute of Technology (AFIT). He worked there until he entered AFIT as a full-time student in May 1988.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.		
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCS/ENG/89D-2			5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION School of Engineering		6b. OFFICE SYMBOL (If applicable) AFIT/ENG		7a. NAME OF MONITORING ORGANIZATION	
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology (AU) Wright-Patterson AFB, Ohio 45433-6583			7b. ADDRESS (City, State, and ZIP Code)		
8a. NAME OF FUNDING / SPONSORING ORGANIZATION AAMRL		8b. OFFICE SYMBOL (If applicable) HEA		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State, and ZIP Code) Wright-Patterson AFB Ohio 45433			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.
			WORK UNIT ACCESSION NO.		
11. TITLE (Include Security Classification) A 3-D Virtual Environment Display System (UNCLASSIFIED)					
12. PERSONAL AUTHOR(S) Robert E. Filer, Captain, USAF					
13a. TYPE OF REPORT MS Thesis		13b. TIME COVERED FROM _____ TO _____		14. DATE OF REPORT (Year, Month, Day) 1989 December	
15. PAGE COUNT 95					
16. SUPPLEMENTARY NOTATION					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Helmet Mounted Displays Computer Graphics		
FIELD	GROUP	SUB-GROUP			
25	03				
12	05				
19. ABSTRACT (Continue on reverse if necessary and identify by block number) Thesis Advisor: Elton P. Amburn, Maj, USAF Instructor					
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT <input type="checkbox"/> UNCLASSIFIED/UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL ELTON P. AMBURN, Instructor			22b. TELEPHONE (Include Area Code) (513) 255-9268		22c. OFFICE SYMBOL AFIT/ENG

UNCLASSIFIED

The design and development of a Virtual Environment Display System is presented. The system is composed of two main parts, a software library to support the development of virtual environment applications and a head-mounted display for viewing the virtual environment.

The software library provides support for numerous input devices including a VPL Data-Glove, Polhemus 3-Space Tracker, Dimension Six Force-Torque Ball, and a joystick. Graphical objects can be displayed in either wireframe or shaded mode. Three dimensional pop-up menus are provided.

The head-mounted display is a fully-enclosed viewing device built using off-the-shelf components. The displays are color LCD televisions and are viewed through wide angle optics. Head position and orientation are tracked using a Polhemus 3-Space Tracker.

UNCLASSIFIED

END